

ulm university universität
uulm

Institut für Angewandte Informationsverarbeitung

Individualisierung statischer und dynamischer Rich Internet Applications unter besonderer Berücksichtigung von Barrierefreiheit

Dissertation
zur Erlangung des Doktorgrades Dr. rer. nat.
der Fakultät für Mathematik und Wirtschaftswissenschaften
der Universität Ulm

vorgelegt von

Norbert Heidenbluth

aus Düsseldorf

2009

Amtierender Dekan:
Prof. Dr. Werner Kratz

Gutachter:
Prof. Dr. Franz Schweiggert
PD Dr. Johannes Mayer
Prof. Dr. Wolfgang Minker

Tag der Promotion:
05. Februar 2010

Inhaltsverzeichnis

1 Einführung **1**

1.1 Motivation 1

1.2 Das "*Big Picture*" 3

1.3 Wissenschaftlicher Beitrag dieser Arbeit 4

2 Barrierefreiheit **5**

2.1 Einführung 5

2.2 Barrierefreiheit in der europäischen Gesetzgebung 7

2.3 Barrierefreiheit in der US-amerikanischen Gesetzgebung 8

2.4 Die W3C Web Accessibility Initiative (WAI) 9

2.5 Grundsätzliches Vorgehen beim Web-Design 9

2.5.1 Web-Design gestern und heute 10

2.5.2 Optimierung auf Endgeräte 11

2.6 Best Practices bei der Entwicklung barrierefreier Webseiten 11

2.6.1 Links und Hypertext 12

2.6.2 Bilder und Grafiken 14

2.6.3 Korrekter Einsatz von Tabellen 15

2.6.4 Einsatz von JavaScript 17

2.6.5 Weitere Prinzipien 23

2.7 Barrierefreiheit und multimediale Inhalte 23

2.7.1 Vor- und Nachteile von Flash in Bezug auf Barrierefreiheit 24

2.7.2 Flash und Screenreader 26

2.8 Der Zusammenhang von Barrierefreiheit und Suchmaschinenoptimierung 26

2.9 Zur Messbarkeit von Barrierefreiheit 27

2.10 Ein erstes Fazit zur Barrierefreiheit im traditionellen Web-Umfeld 27

3 Barrierefreie Authoring Tools **29**

3.1 Motivation 29

3.2 Die Authoring Tool Accessibility Guidelines des W3C 32

3.2.1 Die sieben Richtlinien im Einzelnen und die *Checkpoints* 33

3.2.2 Prioritätsstufen und Erfüllung der Ziele 33

3.2.3 Neuformulierung des Ziels dieses Kapitels 34

3.3 Content-Management-Systeme 34

3.3.1 Trennung von Inhalt und Layout 35

3.3.2 WYSIWYG-Editierfunktion: Vor- und Nachteil zugleich 35

3.3.3 Statische versus dynamische WCM-Systeme 36

3.3.4 Barrierefreiheit und CMS 36

3.3.5 Ein Fazit zur Verwendung von Content-Management-Systemen 37

3.4 Das WebTool-Konzept und dessen prototypische Implementierung 38

3.4.1 Begriffsbestimmungen 39

3.4.2 Anforderungsanalyse 40

3.4.3 Das technische Konzept des WebTools 41

3.4.4 Prototypische Implementierung des WebTools 48

3.4.5 Validierung und Transformation von XML-Daten 53

3.4.6 Diskussion des WebTool-Einsatzes 57

3.5 jLAF 60

3.5.1 Motivation 60

3.5.2 Überblick über das jLAF 61

3.5.3 Konfigurationsparameter 63

3.5.4 Erläuterungen zu ausgewählten Paketen des Frameworks 65

3.5.5 Das Direktivenkonzept: Dynamische Methodenaufrufe via Reflection . . 72

3.5.6 Berücksichtigung der Barrierefreiheit im jLAF 75

3.5.7 Mehrsprachigkeit 75

3.5.8 Diskussion des jLAF 76

4 Rich Internet Applications 79

4.1 Motivation . . . 79

4.1.1 Zum Begriff der “Reichhaltigkeit” . . . 80

4.1.2 Vorteile von RIAs . . . 81

4.1.3 Nachteile von RIAs . . . 82

4.2 Aktuelle RIA-Technologien . . . 83

4.2.1 JavaScript-Bibliotheken . . . 84

4.2.2 Adobe Flex und Adobe AIR . . . 85

4.2.3 Microsoft Silverlight . . . 86

4.2.4 JavaFX . . . 87

4.2.5 Open Laszlo . . . 87

4.3 Usability versus User Experience: Das Dilemma von RIAs . . . 88

4.3.1 Usability . . . 88

4.3.2 User Experience . . . 89

4.3.3 Das Dilemma . . . 89

4.4 Barrierefreiheit und JavaScript-Frameworks . . . 90

4.4.1 Das grundsätzliche Problem . . . 90

4.4.2 Beispiel: Konflikt Hintergrundgrafik / Kontrastreicher Modus . . . 91

4.4.3 Die W3C-Initiative WAI-ARIA . . . 93

4.5 Barrierefreiheit bei Plug-in-basierten RIAs . . . 97

4.5.1 Barrierefreiheit mit Flex (sowie Flash und AIR) . . . 98

4.5.2 Barrierefreiheit in Microsoft Silverlight . . . 102

4.5.3 Barrierefreiheit in JavaFX . . . 102

4.5.4 Barrierefreiheit in OpenLaszlo . . . 103

4.5.5 WAI-ARIA: User Agent Implementation Guide . . . 103

4.5.6 Indizierbarkeit durch Suchmaschinen . . . 103

4.6 Zusammenfassung zu Rich Internet Applications . . . 104

5 HTML-Fallback-Lösungen für statische RIAs 105

5.1 Motivation und Begriffsbestimmungen . . . 105

5.1.1 Statische vs. dynamische RIAs . . . 106

5.1.2 HTML-Fallback-Lösungen und individualisierbare RIAs . . . 107

5.1.3 Zieldefinition . . . 110

5.2 Lösungsansatz auf der Serverseite . . . 111

5.2.1 Das Direktivenkonzept im jLAF-Framework . . . 111
5.2.2 Das RAP-Pattern und Rich Internet Applications . . . 112
5.2.3 Servlet-basierte RAPs . . . 115
5.2.4 RAPs und Remoting Mechansimen . . . 116
5.3 Lösungsansatz auf der (RIA-)Client-Seite . . . 117
5.3.1 Bezug der Inhalte: die Client-Seite des RAP-Patterns . . . 118
5.3.2 Zusammenfassung und Bewertung des RAP-Patterns (client- und serverseitig) . . . 121
5.3.3 Unterstützung von Mehrsprachigkeit: Das MLL-Pattern und dessen Integration im jLAF . . . 122
5.3.4 Visuelle Individualisierbarkeit . . . 127
5.4 Fallback-Lösungen im Umfeld traditioneller und JavaScript-basierter Webseiten 128
5.4.1 Graceful Degradation und Progressive Enhancement . . . 129
5.4.2 Augmentative Authoring . . . 132
5.4.3 Ein zusammenfassendes Beispiel für Graceful Degradation, Progressive Enhancement und Augmentative Authoring . . . 133
5.4.4 Unobstrusive JavaScript . . . 134
5.5 Proof of Concept: Das Webbuch . . . 135
5.5.1 Vorbemerkungen zum Webbuch . . . 135
5.5.2 Serverseitige Implementierung . . . 137
5.5.3 Clientseitige Implementierung . . . 143
5.5.4 Zur Individualisierbarkeit des Webbuchs . . . 146
5.5.5 Zur Generierung der HTML-Fallback-Lösung des Webbuchs . . . 147
5.5.6 Zusammenfassung des Praxisbeispiels . . . 150

6 Strategien zur Laufzeit-Personalisierung dynamischer RIAs 153

6.1 Zur Barrierefreiheit, Individualisier- und Personalisierbarkeit von RIAs . . . 154
6.1.1 Dynamische RIAs und Barrierefreiheit . . . 154
6.1.2 Zum Begriff der Individualisierbarkeit . . . 154
6.1.3 Weitere Begriffe: Personalisierbarkeit und *Customization* . . . 156
6.2 RIAs, Web- und Desktop-Applikationen: Parallelen und Differenzen . . . 156
6.2.1 Traditionelle Online-Anwendungen . . . 156
6.2.2 Zusammenfassende Problembeschreibung . . . 160
6.3 Status Sensitive Components . . . 161
6.3.1 Zustandssensitive Komponenten . . . 163

6.3.2 Das SSC-Pattern . . . 163
6.3.3 Die Hierarchie der `StatusSensitiveComponents` . . . 165
6.3.4 Die Klasse `StatusChangeManager` . . . 166
6.3.5 Die `StatusChangeEvent`-Hierarchie . . . 167
6.3.6 Das SSC-Pattern als UML-Sequenz-Diagramm . . . 167
6.3.7 Diskussion des SSC-Patterns . . . 168
6.4 Privilege Aware Menus (Das PAM-Pattern) . . . 169
6.4.1 Das Konzept hinter dem PAM-Pattern . . . 170
6.4.2 Codieren der Privilegien . . . 172
6.4.3 Das Interface `PrivilegeAwareEntity` . . . 174
6.4.4 Die Klasse `PrivilegeRequirementsInfo` . . . 174
6.4.5 Die Klasse `PrivilegeAwareMenu` . . . 174
6.4.6 Diskussion des PAM-Patterns . . . 176
6.5 Multilanguage Components . . . 177
6.5.1 Querbezug von MLL- und SSC-Pattern . . . 177
6.5.2 Update-Detektion . . . 178
6.6 Zusammenfassung zur Laufzeit-Personalisierung dynamischer RIAs . . . 178

7 Anwendungsbeispiel: SLC on AIR 181
7.1 Überblick über das SLC-System . . . 181
7.1.1 SLC: Vorstellung und Entwicklungshistorie . . . 181
7.1.2 Backendtechnologien in SLC-β . . . 183
7.1.3 Das Web-Frontend in SLC-β . . . 183
7.1.4 SLC on AIR: Die RIA-Version von SLC-β . . . 186
7.2 Anwendung der Pattern auf SLC on AIR . . . 189
7.2.1 Das SSC-Pattern in SLC on AIR . . . 189
7.2.2 Privilege Aware Menus . . . 196
7.2.3 Multilinguale Komponenten . . . 197
7.2.4 Versionskontrolle . . . 197
7.3 Ein Fazit und ein Ausblick zu SLC-β und SLC on AIR . . . 200

8 Zusammenfassung und Ausblick 203
8.1 Zusammenfassung . . . 203
8.2 Ausblick . . . 205

Zusammenfassung (engl.) / Summary **207**

Anhang: Übersicht über JavaScript-Frameworks **209**

Publikationsliste **211**

Danksagungen **213**

Literaturverzeichnis **215**

Verzeichnis der Abbildungen

1.1 Aufbau und Zusammenhänge dieser Arbeit 3

2.1 Mögliches Zusammenspiel der drei Komponenten HTML, CSS und JavaScript . 10
2.2 Gängige Icons zur Darstellung der Sonderbedeutungen von Links 13
2.3 Gegenüberstellung der Funktionsweise traditioneller und AJAX-basierter Webanwendungen (nach [Gar05]) 21

3.1 Grundlegender Aufbau der WebTool-Webseiten 42
3.2 Die vier grundlegenden Schritte der Funktionsweise des WebTools 43
3.3 Screenshot der Begrüßungsseite des IAI (Stand 05/2009) 52
3.4 Screenshot der Begrüßungsseite des IAI (Stand 05/2009) 53
3.5 Ablauf der XML-Transformation im WebTool (hier: IAI-Mitarbeiterseiten) . . . 54
3.6 Unterschiede zwischen uni- und multilingualen XML-Dokumenten 56
3.7 "*Big Picture*" — Die Funktionsweise des jLAF 62
3.8 Gesamtzusammenhang der wichtigsten Pakete des jLAF-Frameworks 66
3.9 Klassenhierarchie im Paket `configurationInfo` 67
3.10 Klassenhierarchie im Paket `directoryEntries` 69
3.11 Abstrakte Klassen und deren individuelle Implementierungen zum Umgang mit XML-Dokumenten 71
3.12 Die Klassen `ReflectionHandler` und `RemoteMethod` 74

4.1 Gegenüberstellung von traditionellen Web-Applikationen und RIAs 84
4.2 Typisches Dialog-Fenster in einer RIA 91
4.3 Eine typische Repräsentierung des GUI-Elements "Schieberegler" 93
4.4 Unterschiedliche Bedeutungen eines simplen Kreises in Flash-Applikationen (nach [Reg05]) 97

5.1 Das ZDF-Politbarometer: Ein Beispiel für eine statische RIA 107
5.2 High-Level-Überblick über das RAP-Pattern 112
5.3 Kapselung des Zugriffs auf den Server via Interfaces 119
5.4 Rückfall auf einfachere Versionen bei *Graceful Degradation* 131
5.5 Sukzessive Erweiterungen bei *Progressive Enhancement* 132
5.6 Screenshot der RIA-Version des Webbuchs (Inhaltsverzeichnis) 136
5.7 Die Implementierungen der Webtool-Entities im Webbook 138
5.8 Klassendiagramm der Entity-Manager 139
5.9 Konkrete Implementierungen der RAPs im Webbuch 140
5.10 Sequenzdiagramm zur Funktionsweise des RAP-Patterns im Webbuch: Bezug von Bildinformationen via Remoting 142
5.11 Sequenzdiagramm zur Funktionsweise des RAP-Patterns im Webbuch: Bezug von Bildinformationen als XML-Dokument (String) 142
5.12 Zusammenhang zwischen den DAO-Klassen auf Client- und Serverseite 146

5.13 Screenshot der Bildergalerie im Webbuch 148

6.1 Desktop- vs. traditionelle Web-Applikationen: Unterschiede/Gemeinsamkeiten . 160
6.2 UML-Klassendiagramm des SSC-Patterns 164
6.3 UML-Sequenzdiagramm zum SSC-Pattern 168
6.4 Laufzeitverhalten des PAM-Patterns 172
6.5 UML-Klassendiagramm des PAM-Patterns 173

7.1 Begrüßungsbildschirm von SLC-β (Web-Version) in Grafik- und Textbrowsern . 185
7.2 Screenshot des Begrüßungsbildschirms von SLC on AIR 188
7.3 Funktionskanon und Status-Voraussetzungen in SLC on AIR 190
7.4 Algorithmus zum Anpassen der Menüstruktur an einen neuen Kontext (Skizze) 192
7.5 Struktur der GUI in SLC on AIR 193
7.6 Reaktion einer online-sensitiven Komponente auf den Abbruch der Online-Verbindung 194
7.7 Reaktion einer login-sensitiven Komoponente auf einen Abmeldevorgang 195
7.8 Screenshot: Hinweis auf die Verfügbarkeit einer neuen Version des RIA-Clients in SLC on AIR 199
7.9 Screenshot: Die verwendete Version von SLC on AIR ist veraltet und kann nicht weiter verwendet werden. 199

Kapitel 1

Einführung

1.1 Motivation

Die Art und Weise, in der heutzutage das *World Wide Web* (WWW) verwendet wird, lässt kaum noch erkennen, dass es ursprünglich lediglich für den Austausch elektronischer Textdokumente gedacht war. Das "Web" im Jahr 2009 ist interaktiv, multimedial und bietet bereits zahlreiche Anwendungen, die als Ersatz für auf dem eigenen Rechner installierte Desktop-Applikationen verwendet werden können[1].

Das hohe Maß an Interaktivität und Multimedialität, das moderne Webauftritte inzwischen bieten, führt das technische Fundament des WWW bisweilen an seine Grenzen, denn dieses hat sich bis heute nicht verändert: noch immer sind HTML und HTTP die tragenden technischen Säulen des weltweiten Netzes, obwohl sie ihrer Intention nach nicht für die Unterstützung der Interaktivität aktueller Webauftritte bestimmt waren. Mit Hilfe von JavaScript können die diesbezüglichen Schwächen von HTML und HTTP weitgehend überbrückt werden, sodass der Benutzer aktueller Webanwendungen alsbald vergisst, dass er nicht mit einer traditionellen Applikation arbeitet, sondern mit "einer Webseite". Einen gänzlich neuen Maßstab in Bezug auf Multimedialität und Interaktivität setzen überdies die "Rich Internet Applications (RIAs)", jene neue Generation von Web-Anwendungen, die sich an klassischen Desktop-Anwendungen orientieren und sich wie diese verwenden lassen. Zur Entwicklung von RIAs konkurrieren zum Zeitpunkt der Erstellung dieser Arbeit mehrere Technologien bzw. Frameworks miteinander. Sie alle werden im weiteren Verlauf vorgestellt.

Bei aller Begeisterung für moderne Internet-Technologien darf jedoch ein Punkt nicht außer Acht gelassen werden: viele Ansätze, um Webauftritte multimedialer und interaktiver zu gestalten, bringen Probleme hinsichtlich der Verwendbarkeit für Menschen mit Einschränkungen mit sich. In der Praxis scheitern RIAs heutzutage insbesondere im Bereich der Barrierefreiheit[2], also dann, wenn sie von Menschen mit körperlichen und/oder geistigen Behinderungen verwendet werden sollen.

An dieser Stelle setzt die vorliegende Arbeit an. Sie führt mit **Kapitel 2** zunächst in den Problembereich der *Barrierefreiheit* ein, gibt einen Überblick über die gesetzliche Lage in der

[1] Als ein Beispiel von vielen sei hier das Bildbearbeitungsprogramm *Photoshop* genannt, das seit Mitte des Jahres 2008 auch in einer vereinfachten Online-Version (*Photoshop Express*, siehe URL `https://www.photoshop.com/`) zur Verfügung steht.

[2] Der Begriff "Barrierefreiheit" fasst die Anforderungen an eine Zugänglichkeit zu Informationen für *alle* Menschen zusammen. Er wird im folgenden Kapitel näher definiert.

europäischen Union (EU) sowie den Vereinigten Staaten von Amerika (USA) und nennt den Status Quo der Barrierefreiheit bei modernen Web-Technologien.

Nach einem Überblick über die *Best Practices* bei der barrierefreien Webseiten-Gestaltung folgt in **Kapitel 3** die Vorstellung des im Rahmen dieser Arbeit entwickelten *WebTools*. Dabei handelt es sich um eine Software-Lösung, mit deren Hilfe kleine[3] (statische) Webauftritte auf einfache Weise barrierefrei konzipiert, unterhalten und auch generiert werden können. Das WebTool orientiert sich strikt an Standards und den zuvor erwähnten *Best Practices*. Es wird als Stand-Alone-Version und als Framework vorgestellt, wobei letzteres als Grundlage für den weiteren Verlauf dient.

Kapitel 4 führt in die Thematik der "Rich Internet Applications (RIAs)" ein und stellt die aktuell am Markt verfügbaren Technologien und Frameworks vor. In diesem Zusammenhang wird dann der Frage, wie RIAs barrierefrei entwickelt werden können, besondere Bedeutung beigemessen. Dazu erfolgen zunächst eine Erläuterung sowie eine Bewertung der diesbezüglich bereits vorhandenen Ansätze. Dies wird schnell deutlich machen, dass eine pragmatische, einfach zu verwendende Lösung noch nicht existiert.

Zu diesen Frage- und Problemstellungen wird in **Kapitel 5** ein eigener Lösungsansatz vorgeschlagen: unter Verwendung und Ergänzung des in Kapitel 3 präsentierten Frameworks wird gezeigt, wie sich aus einer Teilmenge von RIAs ("statischen RIAs", eine Definition dieses Begriffs folgt im entsprechenden Kapitel) automatisiert barrierefreie HTML-Versionen generieren lassen. Diese HTML-Versionen präsentieren sich in der Regel optisch weniger aufwändig; insbesondere sind sie weniger "reichhaltig" (engl.: *rich*) als ihr jeweiliges RIA-Pendant. Weil bei dieser Klasse von RIAs aber der Transport des Inhalts im Vordergrund steht, stellt das keine gravierende Einschränkung dar. Zugleich erlaubt das Framework sowohl bei den RIAs selbst als auch bei den automatisch erstellen HTML-Fallback-Lösungen die Unterstützung mehrerer Sprachen, denn die Beschäftigung mit dem Themenkomplex Barrierefreiheit wird ergeben, dass auch Sprachen Barrieren bedeuten können. Die Integration eines Mehrsprachigkeit-Konzepts in jLAF bzw. die clientseitigen Ergänzungen führen dabei zu einer einfach zu handhabenden, gleichwohl aber gut skalierbaren Lösung.

Fasst man den Begriff der Barrierefreiheit etwas weiter, indem man anstrebt, eine RIA anzubieten, die sich von jedem Benutzer individuell an seine jeweiligen Präferenzen und Notwendigkeiten anpassen lässt, gelangt man in die thematische Sphäre der *Individualisierbarkeit* und *Personalisierbarkeit* von Rich Internet Applications. Eine in diesem Zusammenhang verfolgte Frage ist zum Beispiel, wie sich Navigationsstrukturen einer RIA zur Laufzeit dynamisch an die Berechtigungen des Benutzers anpassen können. Aufgrund der Unterschiede zwischen RIAs und klassischen Web-, aber auch Desktop-Applikationen entstehen in diesem Zusammenhang Fragestellungen, auf die in **Kapitel 6** Antworten gefunden und Lösungsvorschläge unterbreitet werden. Letztere können in Form von Design-Pattern abstrahiert und verallgemeinert werden und stehen fortan für typische Probleme der beschriebenen Art zur Verfügung.

Am Beispiel eines Informationssystems für Studierende der Fakultät für Mathematik und Wirtschaftswissenschaften der Universität Ulm wird in **Kapitel 7** dann gezeigt, wie die vorgestellten Pattern erfolgreich auf den RIA-Client einer realen Web-Anwendung angewendet werden konnten. Eine Zusammenfassung sowie ein Ausblick auf mögliche weitere Entwicklungsschritte runden die Arbeit ab.

[3] Eine Definition des Attributs "klein" erfolgt im Laufe der Arbeit.

1.2 Das “*Big Picture*”

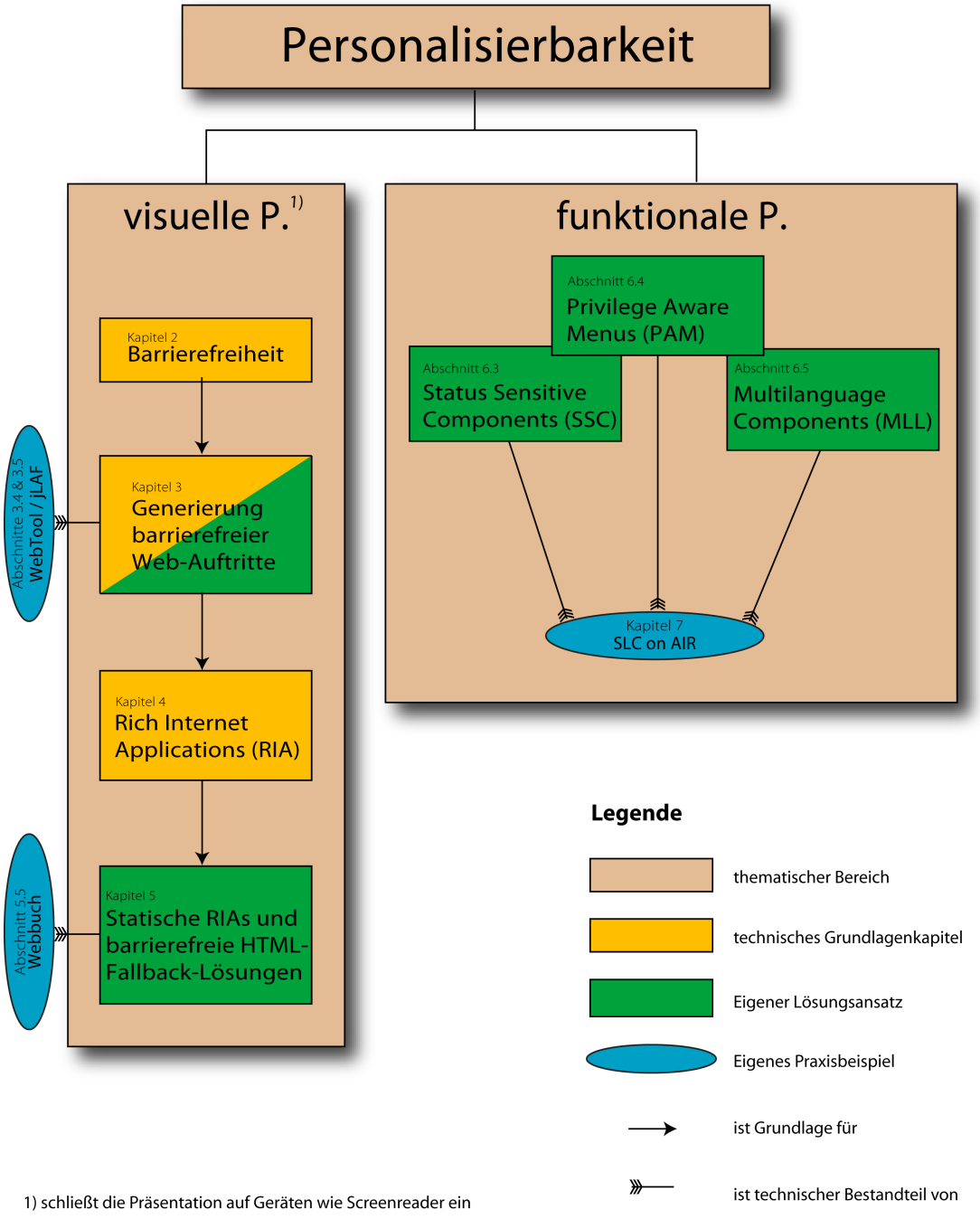


Abbildung 1.1: Aufbau und Zusammenhänge dieser Arbeit

Die in den Abschnitten zuvor verbal ausgedrückte Gliederung wird mit Abbildung 1.1 visualisiert. Die Abbildung zeigt, dass die Arbeit zwei Teilbereiche unterscheidet: “visuelle Personalisierbarkeit” sowie “inhaltlich-funktionale Personalisierbarkeit”. Während der erstgenannte Teilbereich das Thema Barrierefreiheit in den Mittelpunkt der Betrachtungen rückt, liefert der zweite Teilbereich Lösungsansätze für typische Problemstellungen bei RIAs, die sich zur Laufzeit an bestimmte Eigenschaften (z.B. die Berechtigungen des Benutzers) adaptieren müssen.

1.3 Wissenschaftlicher Beitrag dieser Arbeit

Die in Abbildung 1.1 verwendeten Farben geben einen Überblick über diejenigen Stellen, an denen aus der wissenschaftlichen Beschäftigung mit den genannten Fragestellungen eigene Lösungsansätze hervorgegangenen sind (grün eingefärbte Bereiche), sowie über jene Kapitel und Abschnitte, die das Resultat von Literaturrecherchen sind und insoweit den *Status Quo* der jeweiligen Themenbereiche beschreiben (orange eingefärbte Bereiche). Mit den in einem Blauton gefärbten Kapiteln bzw. Abschnitten sind Praxisbeispiele enthalten, die auf der Grundlage der eigenen Lösungsansätze entstanden sind und insoweit deren Praxistauglichkeit beweisen.

Der wissenschaftliche Beitrag lässt sich wie folgt zusammenfassen:

- Die Arbeit beschreibt den *Status Quo* in Bezug auf die barrierefreie Gestaltung von Web-Seiten und Web-Anwendungen (wobei in diesem Zusammenhang letztere auch Rich Internet Applications umfassen).
- Sie legt mit dem WebTool bzw. dem daraus resultierenden Framework "jLAF" eine Software-Lösung vor, um "kleine" Webauftritte barrierefrei und in mehreren Sprachen zu unterhalten, ohne zu diesem Zweck auf ein Content-Management-System zurückgreifen zu müssen.
- Sie ergänzt das (serverseitig einzusetzende) Framework "jLAF" um clientseitig zu verwendende Entwurfsmuster (*Design Pattern*). Die Kombination aus server- und clientseitigen Lösungsvorschlägen lässt sich immer dann einsetzen, wenn statische (informierende) RIAs zu konzipieren und zu unterhalten sind, die
  - in mehreren Sprachen vorliegen,
  - optisch individualisierbar sein und
  - automatisiert in eine barrierefreie HTML-Fallback-Lösung überführt werden

  sollen.
- Mit dem SSC-Pattern (wobei SSC für *Status Sensitive Components* steht) wird ein Entwurfsmuster vorgelegt, das es RIAs erlaubt, ihre Präsentation an "bestimmte" äußere Umstände flexibel zur Laufzeit anzupassen. Dieses Pattern wurde in [HS09b] veröffentlicht und auf der zugehörigen Konferenz einem internationalen Publikum vorgestellt.
- Ergänzend zum SSC-Pattern wird das Konzept der *Privilege Aware Menu Structures* (PAM-Pattern) eingeführt, mit Hilfe dessen RIAs ihre GUI-Elemente (insbesondere die Navigationselemente) zu jeder Zeit an die Berechtigungen (Privilegien) des aktuellen Benutzers adaptieren können. Auch das PAM-Pattern führte zu einer Veröffentlichung ([HS09a]) nebst Präsentation auf einer internationalen Konferenz.
- In die Arbeit eingegangen sind die Resultate zweier betreuter Diplomarbeiten im Studiengang Informatik (siehe [Lučo8] und [Lan08]).

Den technischen Lösungsvorschlägen werden jeweils reale Anwendungen zur Seite gestellt, welche die Praxistauglichkeit der präsentierten Ansätze demonstrieren. Der Fokus dieser Praxisbeispiele liegt dabei auf der Beschreibung der Umsetzung sowie der dabei gesammelten Erfahrungen. Er liegt hingegen nicht auf der Evaluation dieser Beispiele im Hinblick auf dessen Akzeptanz bei den Benutzern und der "*User Experience*" bei der Verwendung der Praxisbeispiele. Eine diesen interessanten Aspekt genauer untersuchende wissenschaftliche Studie ist eine mögliche Fortsetzung dieser Arbeit.

Kapitel 2

Barrierefreiheit

"Accessibility is not just about opening doors—
it is about keeping them open"[Reg05]

2.1 Einführung

Der Vergleich einer durchschnittlichen, auf Kundeninformation ausgerichteten Webseite des Jahres 2009 mit ihren Pendants von vor einigen Jahren lässt erkennen, welchen Design- und *Usability*[1]-Vorsprung HTML-Seiten heutzutage gegenüber ihren Vorgängern haben. Die Verwendung von Cascading Style Sheets (CSS), JavaScript / AJAX[2] und weiteren modernen Technologien haben vergessen lassen, dass HTML-Seiten zunächst einmal gar nicht für aufwändige Layouts und interaktive Darstellungen gedacht waren.

Doch längst nicht alle Benutzer empfinden die zuvor genannten modernen Web-Technologien als hilfreich. Dies kann einerseits daran liegen, dass ein Benutzer im Umgang mit Medien eher pragmatisch geprägt ist und deshalb die reine (mitunter nur textbasierte) Inhaltsvermittlung einer multimedialen Präsentation vorzieht. Auch Ängste und Unwissenheit über die Technologien und die damit verbundenen Gefahren (Sicherheitsrisiken) mögen eine Zurückhaltung oder gar Aversion diesen gegenüber begründen. Benutzer aus dieser Kategorie *möchten* die jeweiligen Technologien nicht verwenden.

Anders verhält es sich hingegen mit der Gruppe von Benutzern, welche die neuen Web-Technologien aus technischen oder gesundheitlichen Gründen nicht verwenden *können*. Technisch einschränken kann beispielsweise die Tatsache, dass sich Inhalte nur mit bestimmten Betriebssystemen und/oder Browsern darstellen lassen ("Plattformabhängigkeit"). Auch die in ländlichen Gebieten oftmals gegebene Nicht-Verfügbarkeit breitbandiger Internetverbindungen stellen für die dortigen Bewohner ein gravierendes technisches Problem im Umgang mit aufwändigen Webseiten dar.

Auf der anderen Seite können körperliche Einschränken wie z.B. Sehbehinderungen oder motorische Störungen das Lesen bzw. Navigieren erschweren oder gar unmöglich machen. Wie

[1] Der englische Begriff *Usability* bezeichnet den komplexen Bereich der Benutzer-Interaktion mit Anwendungen bzw. Webseiten und bezieht die Frage mit ein, wie einfach diese zu verwenden sind. Definition, Übersetzung und Auseinandersetzung mit diesem Begriff folgen auf Seite 88. Aufgrund einer nur ungeschickt möglichen Übersetzung in die deutsche Sprache findet in dieser Arbeit der englische Original-Terminus Verwendung.

[2] AJAX ist die Abkürzung für Asynchronous JavaScript and XML. Der Begriff wird im weiteren Verlauf der Arbeit genauer vorgestellt.

navigiert man auf einer Seite, wenn man die Maus nicht bewegen kann? Wie nimmt man den Inhalt wahr, wenn man den Bildschirm nicht sehen kann?

Eine grobe Kategorisierung der köperlichen Einschränkungen bei der Nutzung des Computers (speziell: des Internets) kann wie folgt aussehen[Utab]:

visuelle Einschränkung: Blindheit, Sehbehinderung, Farbblindheit

akustische Einschränkung: Taubheit, Schwerhörigkeit

motorische Einschränkung: Unvermögen, eine Maus zu verwenden, eingeschränkte motorische Kontrolle, langsames Reaktionsvermögen

kognitive Einschränkung: fehlendes Vermögen, größere Informationsmengen aufzunehmen, Schwierigkeiten mit dem Textverständnis, Lernschwäche, Aufmerksamkeitsdefizit-Syndrom, fehlende Sprachkompetenz

Menschen mit diesen Einschränkungen sind auf das Internet jedoch mindestens genauso angewiesen wie körperlich gesunde Menschen, mitunter sogar deutlich mehr, weil ihnen andere Informationsquellen (z.B. Fernsehen, Tageszeitung) nicht in der Weise zur Verfügung stehen, wie dies bei gesunden Menschen der Fall ist. Insoweit ist es also von besonderer Bedeutung, dass Webseiten auch von behinderten Menschen verwendet und inhaltlich aufgenommen werden können.

Jede der genannten Kategorien von Einschränkungen erfordert von einer Webseite spezielle Vorkehrungen, um für die betroffenen Benutzer verwendbar zu bleiben. Dies kann zum Beispiel bedeuten:

bei visueller Einschränkung: Optimierung des Kontrastes, Verzicht auf Farben, Verwendung großer, gut lesbarer Schriften (Verzicht auf Zierschriften), Optimierung der Seite für Screenreader[3].

bei akustischer Einschränkung: Bereitstellung einer textuellen Version ("Transskript") der multimedialen Inhalte

bei motorischer Einschränkung: Bereitstellung einer Navigationsmöglichkeit mit Hilfe der Tastatur anstelle der Maus (durch sog. *Access Keys*), Verzicht auf Bedienelemente, deren Handhabung mehr als zwei Aktionen zugleich erfordert ("*low bandwith input*[4]")

bei kognitiver Einschränkung: Verzicht auf komplizierte Sprache, ggf. Bereitstellung des Inhalts in zwei Sprachversionen (einmal in "normaler" Sprache, einmal in einer vereinfachten Form)

[3]Screenreader sind spezielle Programme, die den Bildschirminhalt für blinde Menschen vorlesen und Hilfestellung zur Orientierung auf den Seiten bieten.

[4]Mit einer Aktion ist hier beispielsweise ein Mausklick oder ein Tastendruck gemeint. Ein "Drag-'n'-Drop" erfordert demnach *zwei* Aktionen, nämlich das Klicken einer Maustaste, verbunden mit dem zeitgleichen Bewegen der Maus. In [MDBM02] wird in diesem Zusammenhang der Begriff "[Users with] Low Bandwidth Input" gebraucht. Die "niedrige Bandbreite" bezeichnet hierin die Einschränkung des Benutzers, nur ein oder höchstens zwei Signale (Tastendruck, Mausklick, Spracheingabe) zur gleichen Zeit abzusetzen.

2.2 Barrierefreiheit in der europäischen Gesetzgebung

Die Berücksichtigung der genannten Vorkehrungen kommt letztlich allen Benutzern des Internets zugute – nicht nur jenen mit Behinderungen. Mit steigendem Lebensalter lassen oftmals Sehkraft und Auffassungsgabe nach, sodass sich ältere Internet-Nutzer plötzlich Einschränkungen gegenüber sehen, die sie bislang nicht gewohnt waren. Das dieses Kapitel einleitende Zitat bringt diesen Sachverhalt auf den Punkt: Ziel ist es, Informationen für alle Menschen zugänglich (engl.: *accessible*) zu machen und zu erhalten. In Bezug auf behinderte oder ältere Menschen bedeutet dies, dass sie das Internet ohne Hindernisse ("Barrieren") benutzen können.

Den Begriff "Barriere" – bzw. dessen Verwendung im Terminus "Barrierefreiheit" – hat der europäische Gesetzgeber im Behindertengleichstellungsgesetz (BGG) aufgegriffen:

> "Barrierefrei sind bauliche und sonstige Anlagen, Verkehrsmittel, technische Gebrauchsgegenstände, Systeme der Informationsverarbeitung, akustische und visuelle Informationsquellen und Kommunikationseinrichtungen sowie andere gestaltete Lebensbereiche, wenn sie für behinderte Menschen in der allgemein üblichen Weise, ohne besondere Erschwernis und grundsätzlich ohne fremde Hilfe zugänglich und nutzbar sind." (§4, BGG)

Die Erwähnung und Definition des Begriffs "Barrierefreiheit" ist jedoch nur dann von Bedeutung, wenn in Konsequenz auch dessen Berücksichtigung gefordert wird. Für den Bereich der Informationstechnologie wird dies einerseits ebenfalls im BGG geregelt:

> "Träger öffentlicher Gewalt im Sinne des §7 Abs. 1 Satz 1 gestalten ihre Internetauftritte und -angebote sowie die von ihnen zur Verfügung gestellten grafischen Programmoberflächen, die mit Mitteln der Informationstechnik dargestellt werden, nach Maßgabe der nach Satz 2 zu erlassenden Verordnung schrittweise technisch so, dass sie von behinderten Menschen grundsätzlich uneingeschränkt genutzt werden können." (§11, Abschnitt 1, BGG)

Neben dem BGG greift aber andererseits auch die "Barrierefreie Informationstechnik-Verordnung (BITV)". Sie schreibt Richtlinien für die barrierefreie Gestaltung von Webseiten von "Behörden der Bundesverwaltung" (§1: Sachlicher Geltungsbereich) und findet deshalb bereits auf Länderebene keine Anwendung mehr. Stattdessen gelten für die Bundesländer jeweils eigene Bestimmungen. Für die Privatwirtschaft sieht der deutsche Gesetzgeber (noch) keine Verpflichtungen vor, bietet aber in §5 BGG das Instrument der "Zielvereinbarungen", welches im Wesentlichen eine Selbstverpflichtung zur Umsetzung von Barrierefreiheit darstellt.

Die in der Definition des Begriffs "Barrierefreiheit" verwendete Formulierung nach §4 BGG ist nahezu wortwörtlich (bis auf ein wenig "Sprachkosmetik" und Verzicht auf die Nennung von "akustischen und visuellen Informationsquellen") auch im österreichischen Bundes-Behindertengleichstellungsgesetz (BGStG) zu finden (§6, Abschnitt 5). Dies verwundert indes nicht, weil das Behindertengleichstellungsgesetz EU-weit umgesetzt wurde.

Auch die Schweiz (zum Zeitpunkt der Erstellung dieser Arbeit ist sie kein EU-Mitglied) kennt den Begriff "Barrierefreiheit" in ihrer Gesetzgebung. Dort heißt es in der "Verordnung über die Beseitigung von Benachteiligungen von Menschen mit Behinderungen (BehiV)" (151.31, Artikel 10, Abs. 1):

> "Die Information sowie die Kommunikations- und Transaktionsdienstleistungen über das Internet müssen für Sprach-, Hör- und Sehbehinderte sowie motorisch Behinderte zugänglich sein. Zu diesem Zweck müssen die Internetangebote entsprechend den internationalen Informatikstandards, insbesondere den Richtlinien des World Wide Web Konsortiums (W3C) über den Zugang von Internetseiten, und – subsidiär –, entsprechend den nationalen Informatikstandards eingerichtet sein."

Während der EU-Text nicht weiter darauf eingeht, *wie* Barrierefreiheit im Internet erreicht werden kann, schreibt der schweizerische Text bereits "die Richtlinien des W3C" vor (vgl. Abschnitt 2.4). Die deutliche Konkretisierung der technischen Anforderungen ist durchaus sinnvoll gewählt, denn alternative Darstellungsmedien (z.B. Screenreader) sind auf eine ausschließlich am Standard orientierte HTML-Repräsentation des wiederzugebenden Inhalts angewiesen.

2.3 Barrierefreiheit in der US-amerikanischen Gesetzgebung

Einen deutlichen Schritt weiter geht der US-amerikanische Gesetzgeber in "Section 508" des "Rehabilition Act" (zitiert als "Section 508"). Das Äquivalent zum deutschen Begriff "Barrierefreiheit" heißt dort *accessibility*, und mit "Section 508" werden – genau wie in Deutschland bzw. Europa – alle öffentlich-rechtlichen Einrichtungen der USA dazu verpflichtet, alle Informationen sowohl behinderten Mitarbeitern als auch behinderten Bürgern zugänglich zu machen. In 12 Vorschriften nennt der "Rehabilition Act" unter §1194.21 in dem mit "Technical Standards" betitelten Teil B die an Software-Anwendungen und Betriebssysteme gestellten Anforderungen, um als *accessible* (d.h. zugänglich bzw. "barrierefrei") zu gelten. Dabei werden sogar der Gebrauch von Farben und Fortschrittsbalken sowie die Frequenz von blinkendem Inhalt detailliert geregelt, wobei zu beachten ist, dass hier nicht nur auf Internet-Seiten, sondern allgemein auf "Software-Anwendungen" Bezug genommen wird. Für Internet- und Intranetseiten sowie Webapplikationen listet §1194.22 weitere 16 detaillierte Vorschriften auf. Seit dem 21. Juni 2001 müssen alle öffentlich-rechtlichen Webseiten in den USA diesen Anforderungen genügen, andernfalls haben Bürger und Mitarbeiter ein Klagerecht ("Workforce Investment Act"[U.S98]):

> "When developing, procuring, maintaining, or using electronic and information technology, each Federal department or agency, including the United States Postal Service, shall ensure, unless an undue burden would be imposed on the department or agency, that the electronic and information technology allows, regardless of the type of medium of the technology—
>
> i) individuals with disabilities who are Federal employees to have access to and use of information and data that is comparable to the access to and use of the information and data by Federal employees who are not individuals with disabilities; and
>
> (ii) individuals with disabilities who are members of the public seeking information or services from a Federal department or agency to have access to and use of information and data that is comparable to the access to and use of the information and data by such members of the public who are not individuals with disabilities."

2.4 Die W3C Web Accessibility Initiative (WAI)

Während die Gesetzgebung der Schweiz das Konsortium "W3C" explizit nennt, ist dies in den Vorschriften der USA zwar nicht der Fall, gleichwohl entstammen die – teilweise technisch sehr detailliert formulierten – Vorschriften der USA ebenfalls dem W3C. Dieses hat mit seiner *Web Accessibility Initiative* (WAI)[5] Strategien und Richtlinien veröffentlicht, auf welche die amerikanischen Vorschriften (und entsprechende Gesetze in vielen anderen Ländern) zurückzuführen sind.

Zum Zeitpunkt der Erstellung dieser Arbeit liegen mit den WCAG ("Web Content Accessibility Guidelines") Richtlinien zur Erstellung barrierefreien Web-Inhalts des W3C vor ([W3C08b], aktuell in der Version 2.0 vom 11. Dezember 2008. Die Richtlinien basieren auf der Maßgabe, dass jeder, der das Internet verwendet, Inhalt vorfinden soll, welcher die folgenden vier Eigenschaften (im Originaldokument als Prinzipien [*principle*] benannt und hier sinngemäß mit eigenen Worten übersetzt) aufweist:

wahrnehmbar (*perceivable*): Der Inhalt muss in einer für den Benutzer wahrnehmbaren Art präsentiert werden, d.h. der Inhalt darf nicht allen Sinnen (zugleich) verschlossen bleiben.

bedienbar (*operable*): Die Benutzerschnittstelle sowie die Navigation müssen für den jeweiligen Benutzer bedienbar sein.

verständlich (*understandable*): Sowohl der präsentierte Inhalt als auch die Bedienung der Benutzerschnittstelle müssen für den Benutzer verständlich sein.

robust (*robust*): Der Inhalt muss sich auf einer Vielzahl unterschiedlicher Technologien (z.B. Browsern) darstellen lassen und soll auch mit zukünftigen Browsern funktionieren.

Die Richtlinien gründen auf der Überzeugung, dass, wenn bereits eine dieser Eigenschaften nicht erfüllt ist, die jeweilige Seite nicht mehr für alle Menschen zugänglich (*accessible*) ist.

2.5 Grundsätzliches Vorgehen beim Web-Design

Die Gestaltung von Webseiten beruht aktuell auf drei Komponenten:

1. HTML,
2. CSS und
3. JavaScript,

die jedoch nicht notwendigerweise stets gemeinsam verwendet werden müssen.

Abbildung 2.1 zeigt die Aufgaben dieser drei Komponenten und deren mögliche Beziehungen untereinander. Die wichtigste Bedeutung kommt der HTML-Komponente zu, weil (nur) sie den gesamten Inhalt bzw. dessen semantische Struktur repräsentiert. CSS ist optional als Präsentationskomponente (d.h. Layoutkomponente) zu verwenden. Interaktionspotential (d.h.

[5] http://www.w3.org/WAI

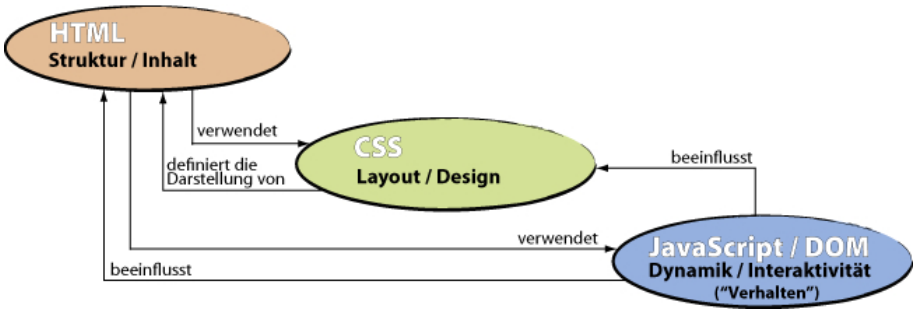


Abbildung 2.1: Mögliches Zusammenspiel der drei Komponenten HTML, CSS und JavaScript

Dynamik bzw. "Verhalten") lässt sich mit der dritte Komponente erreichen: JavaScript. Diese steht ebenfalls optional zur Verfügung. Wie aus der Abbildung ersichtlich, ist JavaScript die einzige Komponente, mit der sich die beiden anderen beeinflussen lassen – insbesondere auch aufgrund von Benutzerinteraktionen. Dies kann je nach Einsatz zu einer Verbesserung bei Individualisierbarkeit und Barrierefreiheit führen, aber – bei nicht sorgfältiger Verwendung – auch einen gegenteiligen Effekt haben. Die Vor- und Nachteile von JavaScript, gerade im Hinblick auf Usability und Barrierefreiheit, sind in den Abschnitten 2.6.4 und 4.4 Gegenstand einer ausführlichen Diskussion. Zudem wird die in Abbildung 2.1 visualisierte Trennung von Inhalt, Layout und Verhalten nochmals im Zusammenhang mit den Paradigmen "Graceful Degradation" und "Progressive Enhancement" (vgl. Abschnitte 5.4.1ff. auf den Seiten 129ff.) aufgegriffen.

2.5.1 Web-Design gestern und heute

Müsste man das Wesentliche, das es zu beachten gilt, um eine barrierefreie Webseite zu erstellen, in einem Satz zusammenfassen, so würde dieser wohl in etwa lauten "*Trenne das Layout vom Inhalt und orientiere dich dabei an den Web-Standards HTML (für den Inhalt) und CSS (für das Layout)*". So banal diese Forderung auf den ersten Blick klingen mag: auch zum Zeitpunkt der Erstellung dieser Arbeit weist das Web zahlreiche Seiten auf, die diese Empfehlung nicht erfüllen. Ein besonders prominentes Beispiel für einen Verstoß gegen diese Empfehlung ist hierbei die Verwendung von Tabellen als Layout-Mittel (vgl. dazu Abschnitt 2.6.3). Durch eine solche Zweckentfremdung der Tabellen wird die Trennung von Layout und Inhalt aufgehoben, was sich – wie später noch ausgeführt wird – sehr negativ auf die Barrierefreiheit auswirken kann.

Die Empfehlung, Layout und Inhalt voneinander zu trennen und sich dementsprechend strikt an HTML- und CSS-Standards zu halten, hat in der Vergangenheit zu Problemen geführt: längst nicht jeder Browser agierte standardkonform. Um dennoch alle gängigen Browser verwenden zu können, wurden in die Seiten Fallunterscheidungen aufgenommen, die den verwendeten Browsertyp erkannten und dessen spezifische Inkompatibilitäten umgingen. Dieses "Umgehen" bestand oftmals aus der Verwendung von HTML-Konstrukten, die in einer Form benutzt wurden, für die sie nicht unmittelbar bestimmt waren. Deshalb hat sich dafür der Begriff "Browserhack" etabliert. An Web-Standards orientierten sich diese "Hacks" in der Regel nicht mehr, sodass sie im Hinblick auf Barrierefreiheit sehr fragwürdig erscheinen. Aktuelle Browser verhalten sich nunmehr jedoch sowohl bei HTML als auch bei CSS standardkonform, sodass die Ära der "Browserhacks" überwunden ist. (Das Problemfeld der Browserinkompatibilitäten wird ebenfalls im Zusammenhang mit "Graceful Degradation" und "Progressive Enhancement" nochmals aufgegriffen.)

Die reine Einhaltung von Web-Standards genügt aber noch nicht, um eine Webseite zu gestalten, die für alle Ausgabegeräte, Anwendungsszenarien und auch Benutzer gleichermaßen gut geeignet ist. Aufgrund der vielen Endgeräte, die mittlerweile als Zugangsmedium zum Internet genutzt werden können, kommt der Frage der Bildschirmgröße (Auflösung) eine wichtige Bedeutung zu. Mit CSS lassen sich sowohl *absolute* als auch *relative* Layouts realisieren, d.h. Größenangaben können in einer festen Pixelgröße oder in relativen Maßen – bezogen auf den verfügbaren Platz – angegeben werden. Nur die konsequente Verwendung von relativen Angaben erlaubt jedoch eine weitgehend reibungslose Übertragbarkeit der Webseiten auf die vielfältigen Ausgabegeräte (mit ihren ebenso vielfältigen Auflösungsvariationen). Absolute Größen sind nur dann zuverlässig zu verwenden, wenn man sicher davon ausgehen kann, dass die Seite mit einer bestimmten Auflösung (oder innerhalb bestimmter Grenzen von Auflösungen) angezeigt werden soll. Mit Blick auf die Vielfalt der Endgeräte – vom Mobiltelefon bis hin zum 30-Zoll-Monitor – sollten die Festlegung auf bestimmte Auflösungen sowie die Verwendung von absoluten Layouts jedoch unterlassen werden.

2.5.2 Optimierung auf Endgeräte

Ungeachtet der Ausführungen im voranstehenden Abschnitt sind Szenarien denkbar, in denen eine Webseite auf bestimmte Ausgabemedien (z.B. Mobiltelefone mit einer bekannten Auflösung) optimiert werden soll. Hier bietet sich die Verwendung der dritten Komponente aus Abbildung 2.1 an: JavaScript. Damit ist es möglich, dynamisch beim Aufbau der Seite auf dem Endgerät die aktuelle Auflösung zu ermitteln und abhängig davon eine geeignete CSS-Datei zu verwenden. So kann es beispielsweise eine Version für "normale" Bildschirme geben sowie eine weitere, die für die recht kleinen Displays von Mobiltelefonen optimiert ist.

Diese einführende Übersicht über die drei Komponenten im Web-Design war soweit noch allgemein gehalten und hatte nicht die Barrierefreiheit im Fokus. Fragen, Empfehlungen sowie Probleme und deren Lösungen bei der Gestaltung barrierefreier Webseiten stehen jedoch im Mittelpunkt des folgenden Abschnitts.

2.6 Best Practices bei der Entwicklung barrierefreier Webseiten

Die genannten Richtlinien des W3C geben keine konkreten Empfehlungen, wie ihre Umsetzung erreicht werden kann. Ein wesentlicher Aspekt beim Bestreben, barrierefreie Webseiten zu erstellen, ist die strikte Beachtung des HTML- und des CSS-Standards. Dies allein ist jedoch noch nicht ausreichend. Aus diesem Grund existieren große Mengen von weiteren Empfehlungen für die Umsetzung der soeben vorgestellten Prinzipien. Manche von ihnen werden kontrovers diskutiert, andere haben sich als *de facto*-Standards etabliert ("*Best Practices*"). Die Anwendung dieser Empfehlungen bedeutet in den allermeisten Fällen keinen oder nur einen vernachlässigbar kleinen Aufwand bei der Entwicklung der Webseiten. Zudem wird durch die Beachtung der genannten Prinzipen das Layout der Seite(n) nicht verändert, sodass für den durchschnittlichen Benutzerkreis (also jenen ohne Einschränkungen) kein Unterschied erkennbar ist.

Um im weiteren Verlauf dieser Arbeit ein Fundament zur Verfügung zu haben, auf dem sich die weiteren Schritte begründen und aufbauen lassen, erscheint es sinnvoll, die wichtigsten (und auch etabliertesten) dieser Empfehlungen zunächst geordnet zusammenzufassen und zu begründen. Dies wird – thematisch sortiert – in den nachfolgenden Abschnitten gemacht, und

auf den dort vorgestellten Empfehlungen basieren dann auch die praktischen Implementierungen, die im Rahmen dieser Arbeit entstanden sind. Die Zusammenstellung der Empfehlungen ist insbesondere auch eng an die *Principles of Accessible Design* der WebAIM-Initiative[Utac] angelehnt, weil diese federführend im Umfeld von *Accessibility*-Fragen und -Standardisierungen agiert.

2.6.1 Links und Hypertext

Verweise (Links) erfordern sowohl für sehbehinderte Menschen als auch für Benutzer, die nicht mit der Maus navigieren (können), besondere Vorkehrungen, um verwendbar zu bleiben.

Wird anstelle der Maus die Tastatur zur Navigation verwendet, so dient eine bestimmte Taste (in der Regel wird es sich hierbei um die Tabulator-Taste handeln) zum Sprung von Link zu Link und eine weitere zum Auswählen des momentan aktiven Links (in der Regel "Enter" bzw. "Return"). Sofern die Links ausschließlich auf HTML basieren, ist dies unmittelbar möglich und damit (in diesem Kontext) barrierefrei. Allerdings werden Links auch oftmals in Zusammenhang mit JavaScript oder gar mit Flash-Technologie verwendet. Im Abschnitt 2.6.4 werden die Probleme beim Einsatz von JavaScript detailliert erläutert, und Abschnitt 2.7 widmet sich dem Umgang mit Flash-basierten Inhalten. Deshalb sei an dieser Stelle nur kurz erwähnt, dass Links, die nicht in reiner HTML-Form Verwendung finden, für Benutzer, die auf die Tastatur als Eingabemedium angewiesen sind, nicht unmittelbar (d.h. allenfalls nur durch besondere Vorkehrungen) zugänglich sind. Weiterführende Erklärungen dazu folgen in den genannten Abschnitten.

Mit einer anderen Problematik im Umgang mit Links sieht man sich konfrontiert, wenn Links für den Einsatz von Screenreadern angepasst werden sollen. Ein Gespür für die Problematik bekommt man bereits, wenn man sich überlegt, wie Links (d.h. Inhalte einer Webseite, die zugleich als Verweis auf andere Seiten dienen) wahrgenommen werden können: während normal sehende Menschen einen Link in der Regel dadurch erkennen, dass er farblich hervorgehoben und/oder unterstrichen ist, kann ein Screenreader diese Form der Darstellung nicht übernehmen. Stattdessen wird (je nach eingesetztem Screenreader) entweder eine weibliche anstelle einer männlichen Stimme verwendet oder (seitens des Screenreaders) automatisch das Wort "Link" vor den vorzulesenden Text (d.h. vor das Ziel des Links) gestellt. Genau wie zuvor beschrieben, dient beim Einsatz von Screenreader auch eine Navigationstaste (in der Regel wohl "Tab") zum Springen von Link zu Link, wobei nun besonders zu beachten ist, dass der sehbehinderte Benutzer auf diese Weise nicht des Kontextes, in dem sich der Link befindet, gewahr wird und deshalb der Link-Text – also jener Text, der als Link hervorgehoben dargestellt wird – selber bereits aussagekräftig sein muss. Dies bedeutet insbesondere, dass auf Phrasen wie "Hier klicken" und ähnliches zu verzichten ist, weil diese keinerlei Information über das eigentliche Ziel des jeweiligen Links liefern. Stattdessen sollte der Link-Text allein bereits deutlich zum Ausdruck bringen, auf was der Link verweist ("Klicken Sie hier für den aktuellen Wetterbericht").

Aber auch diese Darstellung eines Links ist noch nicht optimal, denn wie zuvor erwähnt, geben einige Screenreader das Wort "Link" automatisch mit aus, sodass im Beispiel die Ausgabe lauten würde: "Link zu Klicken Sie hier für den aktuellen Wetterbericht". Dies ist zwar verständlich, aber unschön. Deshalb sollte auf das "Klicken Sie hier" (sowie ähnliche Phrasen, die keine weiteren Informationen über einen Link beisteuern) im Sinne der besseren Verständlichkeit für Benutzer von Screenreadern vollständig verzichtet werden. Im Beispiel wäre der Link-Text "Aktueller Wetterbericht" (vom Screenreader selbstständig expandiert zu "Link zu

Aktueller Wetterbericht") bereits ausreichend. Aus demselben Grund sollte der Link-Text andererseits aber auch nicht zu lang sein und insbesondere keine für Menschen unlesbaren Inhalte aufweisen[6].

Links verweisen aber nicht nur auf andere HTML-Seiten sondern oftmals auch auf Dateien, die in einem anderen Format vorliegen (pdf, jpg, doc, etc.). Diese Formate sind für Screenreader nicht darstellbar, und deshalb sollte bereits der Link-Text darauf hinweisen, dass das Linkziel ein Fremdformat (d.h. kein HTML-Format) darstellt. Technisch ist es möglich, einen solchen Link automatisch durch das jeweilige Ausgabegerät erkennen zu lassen, und somit kann das Gerät den Link um diese Information ergänzen. Einen solchen Ansatz verfolgt Michael Bolin mit der Erweiterung *TargetAlert* für den Webbrowser *Firefox* (siehe [Bol04]). Die Unterstützung derartiger Technologien auf den Zielrechnern der durchschnittlichen Internet-Benutzer kann zum Zeitpunkt der Erstellung dieser Arbeit jedoch nicht vorausgesetzt werden, weshalb derzeit noch der Autor einer Webseite dafür Sorge tragen muss, dass ein Link die Information, dass er auf ein Fremdformat verweist, enthält.

Die Empfehlung, spezielle Links (also solche, die nicht nur auf eine andere HTML-Seite führen), besonders zu kennzeichnen, lässt sich aber noch weiter führen: Links können in einem neuen Fenster geöffnet werden oder beim Einsatz von HTML-Frames einen anderen Frame ansprechen. Sie können auf verschlüsselte Seiten verweisen (HTTPS) oder den eigenen Internetauftritt verlassen und auf sogenannte "externe Seiten" führen.[7] Ferner ist es möglich, dass sie noch weitere, speziellere Bedeutungen haben bzw. Aktionen auslösen. Wie im Absatz zuvor beschrieben, ist es durchaus möglich, dass derartige Sonderfunktionen automatisch vom Ausgabegerät (z.B. Browser) erkannt und ausgegeben werden. In der Praxis hat sich dieses Verhalten jedoch noch nicht etabliert. Deshalb sollten Autoren von Webseiten diese Sonderbedeutungen im Link-Text oder durch ein zusätzliches kleines grafisches Symbol ausdrücken. Beispiele für etablierte (jedoch nicht standardisierte) Icons gibt Abbildung 2.2.



Abbildung 2.2: Gängige Icons zur Darstellung der Sonderbedeutungen von Links

Hervorzuheben ist an dieser Stelle, dass, wenn die Empfehlungen, die aus der vorangegangenen Diskussion resultieren, umgesetzt werden, dies für normal sehende Benutzer mit keinerlei Einschränkungen oder Nachteilen in irgendeiner Form verbunden ist. Hingegen könnte aus der Empfehlung, grafische Icons zu verwenden, gefolgert werden, dass dies der beabsichtigten Barrierefreiheit zuwiderläuft, denn Screenreader sind nicht in der Lage, Grafiken und Bilder unmittelbar auszugeben. Hierzu sei angemerkt, dass sich Barrierefreiheit nicht nur auf sehbehinderte Benutzer beschränkt. Abgesehen davon kann die Verwendung von Grafiken durchaus

[6] Damit ist zum Beispiel eine URL gemeint, die verschiedene Parameter für ein Skript enthält, wie z.B. `http://www.amazon.de/exec/obidos/tg/feature/-/1000152033/`. Dieser Link verweist z.B. beim Online-Buchhändler *Amazon* auf die Produktbeschreibung eines Fernsehers, ist für den Menschen aber gänzlich unverständlich.

[7] Der Wunsch, derartige Links besonders zu kennzeichnen, begründet sich in der Regel durch die Rechtsprechung, welche dem Betreiber einer Webseite die Haftung auch für verlinkte (externe) Inhalte auferlegt. Durch die Kennzeichnung eines Links als externen Link nebst einer entsprechenden Formulierung im Impressum der Seite versuchen viele Autoren von Webseiten, sich zu exkulpieren. Die Frage, ob sich die Haftung für externe Inhalte auf diese Weise vermeiden lässt, ist in der (deutschen) Rechtsprechung zum Zeitpunkt der Erstellung dieser Arbeit jedoch noch strittig.

auch in einer Form erfolgen, dass alternative Ausgabegeräte mit den Grafiken umgehen können. Hierzu sei auf den nachfolgenden Abschnitt 2.6.2 verwiesen.

Für Benutzer von Screenreadern ist noch eine weitere mögliche Barriere zu beachten: die klare Trennung von Links, die unmittelbar nebeneinander platziert sind. Dies kann zum Beispiel ein alphabetischer Index (d.h. eine Auflistung der Buchstaben "A" bis "Z") sein. Screenreader, die Links vom normalen Text durch den Einsatz einer weiblichen anstelle einer männlichen Stimme unterscheidbar machen, erkennen, dass zwischen den einzelnen Links ("A", "B", etc.) kein weiterer Text steht, und verbleiben in der weiblichen Stimme. Daher ist eine klare Abgrenzung der Links untereinander (beispielsweise die Unterscheidung, ob es sich um die Aufzählung "A", "B" oder um "AB" handelt) für den Benutzer nicht mehr oder nur sehr schwer möglich. Um dieses Problem zu umgehen, empfiehlt sich die Einbettung der Links in eine HTML-Liste. Die einzelnen Elemente einer HTML-Liste werden von Screenreadern in der Regel durch eine längere Pause voneinander unterschieden, was deren Verständlichkeit für den Benutzer erhöht. Alternativ ist es möglich, Trennzeichen – z.B. das Pipe-Symbol ("|") – zwischen den einzelnen Links zu notieren, wobei dieses Vorgehen eher als *Workaround* denn als Problemlösung angesehen werden sollte.

Der Vollständigkeit halber sei noch eine weitere Empfehlung genannt, die sich auf alle normalsichtigen Benutzer bezieht: die Erkennbarkeit eines Links "auf den ersten Blick" (z.B. durch farbliche Hervorhebung und/oder Unterstreichung) sollte im Hinblick auf die Verständlichkeit der Seite stets gewährleistet sein. Oder anders formuliert: auf Links, die sich nicht vom normalen Fließtext abheben und somit nicht unmittelbar als Links zu erkennen sind, sollte verzichtet werden.

2.6.2 Bilder und Grafiken

Alternative Ausgabegeräte (z.B. Braille-Display, Screenreader, Textbrowser und ähnliche Hilfsmittel) können Bilder und Grafiken (nachstehend als "grafische Elemente" bezeichnet) nicht darstellen. Andererseits können diese Geräte aber den durch die grafischen Elemente transportierten Inhalt in der Regel auch nicht analysieren, um ihn selbstständig in textuelle Form zu übertragen. Umso wichtiger ist es, den Einsatz von Bildern und Grafiken so zu gestalten, dass auf sie verzichtet werden kann, ohne dass dies mit einem Informationsverlust einhergeht.

Zu diesem Zweck sieht der HTML-Standard (siehe [W3C99]) vor, dass Bilder und Grafiken (notiert durch das `img`-Tag) mit einem Alternativtext versehen werden, der immer dann ausgegeben wird, wenn eine Darstellung des Elements selber nicht möglich ist. Der gewünschte Alternativtext wird im Attribut `alt` des jeweiligen `img`-Tags notiert. Allerdings bezieht sich der Begriff "Alternativtext" auf mehr als nur den Inhalt des entsprechenden Attributs. So ist es insbesondere denkbar, dass der eine Grafik umfließende Text auch eine Beschreibung dieser Grafik enthält. Wenn dann die Grafik nicht wahrgenommen werden kann, dann gehen durch den beschreibenden (Fließ-)Text keine wesentliche Informationen verloren. Generell lautet die Empfehlung in [Utac], dass

- jedes grafische Element ein `alt`-Attribut besitzen muss – auch, wenn bewusst kein Alternativtext angegeben werden soll (in diesen Fällen bleibt das `alt`-Attribut leer: `alt=""`) – und
- der Alternativtext sowohl die Funktion als auch den Inhalt des grafischen Elements wiedergeben muss und möglichst kurz gehalten werden soll.

Der zweite Punkt der vorstehenden Aufzählung differenziert zwischen der *Funktion* und dem *Inhalt* eines grafischen Elements und bezieht sich damit auf den Zweck, dem die Verwendung dieses Elements dienen soll. Neben Funktion und Inhalt lässt sich auch noch ein dritter Zweck nennen: jener der *Dekoration*. Diese drei Zwecke bzw. die sich daraus ergebenden Charakterisierungen *funktional*, *inhaltlich* und *dekorativ* können wie folgt unterschieden werden:

funktional: Werden grafische Elemente zur Navigation eingesetzt (z.B. als Link), so haben sie eine funktionale Aufgabe. In diesem Fall gelten für den Text innerhalb des `alt`-Attributs die zuvor im Zusammenhang mit der Aufbereitung von Links für Screenreader erläuterten *best practices* analog.

inhaltlich: Sofern der Inhalt des grafischen Elements einen wesentlichen inhaltlichen Beitrag zur Seite leistet (z.B. ein Diagramm), muss dieser Inhalt in alternativer (Text-)Form zur Verfügung gestellt werden. Da der Text innerhalb des `alt`-Attributs andererseits aber auch kurz gehalten werden sollte (damit der Benutzer von Screenreadern nicht zu sehr vom eigentlichen Fließtext ablenkt), eignet sich das `alt`-Attribut oftmals nicht, um die textuelle Version des Bildinhalts aufzunehmen. Für diesen Fall steht in HTML für das `<img>`-Tag das Attribut `longdesc` zur Verfügung. In dieses kann ein Link eingebettet werden, welcher auf eine separate Seite verweist. Diese separate Seite enthält dann eine ausführliche textuelle Umschreibung des ursprünglichen Bildes. Der Vorteil dieser Vorgehensweise liegt darin, dass nun der Benutzer selbstständig entscheiden kann, ob er dem Link auf die Seite folgt oder nicht. Unterlässt er es, so kommt es zu keiner Unterbrechung bzw. Ablenkung im Informationsfluss der ursprünglichen Seite.

dekorativ: Dekorative grafische Elemente können zum Beispiel Trennlinien oder Ornamente sein. Sie haben keine wesentliche Funktion (abgesehen von der "Funktion", die Seite optisch aufzuwerten) und vermitteln auch keinen Inhalt. Deshalb können sie von alternativen Ausgabegeräten ignoriert werden, ohne dass dies einen Verlust an Inhalt zur Folge hätte. Man könnte daher das `alt`-Tag –wie oben beschrieben– leer lassen, dann aber würde das Bild dennoch als solches vom Screenreader erkannt und unnötigerweise "erwähnt". Um dekorative grafische Elemente für alternative Darstellungsformen unsichtbar zu machen, sollten diese stattdessen via CSS als Hintergrundbild eingebettet werden. Dann sind diese Hintergrundbilder nicht in der HTML-Seite selbst enthalten und werden deshalb von Screenreadern auch nicht berücksichtigt.

Natürlich sind auch Mischformen möglich, insbesondere können grafische Elemente sowohl einen funktionalen als auch einen inhaltlichen Zweck erfüllen. Grundsätzlich gilt – wie bereits im Abschnitt über Links konstatiert – dass die Bemühungen um einen barrierefreien Einsatz von grafischen Elementen *allen* Benutzern zugute kommen. Bei strikter Trennung von Layout und Inhalt bietet sich die Möglichkeit, z.B. eine für den Druck optimierte Darstellung der Seite bereitzustellen oder dem Benutzer verschiedene Layout-Alternativen (z.B. "normal", "besonders kontrastreich", "Verwendung großer Schriften" etc.) anzubieten. Durch diese Trennung lässt sich ein semantisch saubererer HTML-Quelltext erreichen, welcher wiederum auch einfacher ausgewertet werden kann, beispielsweise durch Suchmaschinen (siehe dazu auch Abschnitt 2.8).

2.6.3 Korrekter Einsatz von Tabellen

Eine besondere Bedeutung im Hinblick auf Barrierefreiheit kommt der Gestaltung von Tabellen zu. Tabellen sind in der Regel nicht linear (d.h. "von oben nach unten") zu lesen, sondern

erfordern eine Interpretation ihrer Darstellung, die nicht-sehenden Menschen nur sehr schwer – und bei falscher Gestaltung eventuell auch gar nicht – vermittelt werden kann. Grundsätzlich kann der Einsatz von HTML-Tabellen wie folgt unterschieden werden:

Daten-Tabellen: Sie dienen der Aufbereitung von Daten (Informationen) im typischen Tabellen- bzw. Matrix-Stil; sie sind daran zu erkennen, dass sie eine Zeilen- oder Spaltenbeschriftung (oder beides) aufweisen.

Layout-Tabellen: Sie dienen als reines Gestaltungmittel von kompletten Webseiten und/oder Teilen davon; sie besitzen keine Spalten- und Zeilenbeschriftungen.

2.6.3.1 Daten-Tabellen

Daten-Tabellen dienen der Aufbereitung von Informationen in tabellarischer Form, d.h. als Matrix mit m Zeilen und n Spalten. Je größer m und n sind, desto schwieriger ist es für Benutzer von Screenreadern, den Inhalt zu erfassen, denn Screenreader lesen den Inhalt von Tabellen in der Regel zeilenweise Spalte für Spalte vor. Es kann bei einer großen Spalten- bzw. Zeilenanzahl daher schnell passieren, dass der Benutzer nicht mehr weiß, welche Bedeutung die Spalte bzw. Zeile, in der er sich mit der aktuellen Zelle momentan befindet, überhaupt besitzt.

Damit diese Informationen jederzeit abgerufen werden können, ist die korrekte Beschriftung von Zeilen und Spalten einer Datentabelle unabdingbar. HTML kennt zu diesem Zweck das Tag `<th>` (*table header*), das die enthaltene Zelle als Beschriftungszelle auszeichnet. Ferner kennt HTML das Tag `<scope>`, das einen Bezug zwischen einer Datenzelle und der zugehörigen Kopfzelle herstellt. Dieses Tag gehört zu dem Satz von HTML-Sprachelementen, die eigens für die Verwendung in Screenreadern in den HTML-Standard aufgenommen wurden. Eine ganz ähnliche Funktion erfüllt das Attribut `<id>`, welches ebenfalls einen Bezug zwischen Daten- und Kopfzellen herstellt und für den Einsatz in Daten-Tabellen mit einem sehr komplexen Aufbau vorgesehen ist. Ein weiteres Attribut, das für alternative Ausgabegeräte vorgesehen ist, stellt das Attribut `<summary>` dar. Es gehört zum Tabellen-Tag (`<table>`) und kann dazu verwendet werden, die wesentliche Aussage einer Tabelle in einem kompakten Kurztext zusammenzufassen – analog zum Alternativ-Text bei der Verwendung von Bildern (vgl. vorstehender Abschnitt). Weil es als Attribut des `<table>`-Tags noch vor der eigentlichen Ausgabe des Tabelleninhalts interpretiert wird, kann der Benutzer eines Screenreaders bereits vor der Ausgabe der eigentlichen Tabelle auf diejenigen Informationen der Tabelle aufmerksam gemacht werden, auf die es ankommt.

Wenngleich das `<summary>`-Attribut für den Einsatz durch Screenreader vorgesehen ist, so hilft eine solche Zusammenfassung nicht nur sehbehinderten Menschen, sondern beispielsweise auch jenen mit kognitiven Schwächen. Wie früher in diesem Kapitel bereits dargestellt, bezieht der Begriff Barrierefreiheit auch diese Menschen mit ein, und eine Zusammenfassung des wesentlichen Inhalts komplexer Sachverhalte stellt für sie eine Hilfe dar.

Weiterführende Informationen zu den hier erwähnten (und weiteren) HTML-Tags für alternative Ausgabemedien liefert zum Beispiel die HTML-Referenz oder [Mü07].

2.6.3.2 Layout-Tabellen

Wenn Tabellen als reines Gestaltungsmittel von HTML-Seiten zum Einsatz kommen sollen, so ist dies im Hinblick auf Barrierefreiheit als sehr problematisch zu werten. Vor der Erläuterung

der auftretenden Probleme soll zunächst ein Begriff eingeführt werden:

Definition – Linearisierung: Unter der *Linearisierung* einer (Web-)Seite bzw. ihrer *linearisierten Form* versteht man deren Darstellung, wenn jegliche Formatierung sowie insbesondere auch die HTML-Tags unberücksichtigt bleiben (entfernt werden); lediglich der zwischen den HTML-Tags enthaltene Inhalt der Seite bleibt übrig.

Der linearisierten Form einer Webseite kommt für die Beurteilung deren Barrierefreiheit eine wichtige Bedeutung zu. Denn die linearisierte Form entspricht genau dem, was Screenreader ausgeben. Somit muss also stets dafür Sorge getragen werden, dass die inhaltliche Logik (Struktur) einer Seite auch in der linearisierten Version erhalten bleibt. Dies zu gewährleisten ist beim Einsatz von Tabellen als Gestaltungsmittel jedoch nahezu unmöglich, denn die Linearisierung eines HTML-Quelltextes, in dem Tabellen enthalten sind, führt dazu, dass der Inhalt der Tabellen in genau der Reihenfolge ausgegeben bzw. vorgelesen wird, in welcher er im Quelltext auftritt. Deshalb weisen die linearisierten Versionen dieser Seiten in der Regel eine für alternative Ausgabegeräte nicht mehr nachzuvollziehende Struktur auf.

Eine Möglichkeit, diesem Problem zu begegenen, ist der völlige Verzicht von HTML-Tabellen als reines Gestaltungsmittel. Alternativ können anstelle der Tabellen Stylesheets (CSS) zum Einsatz kommen, welche – wie bereits beschrieben – eine vollständige Trennung von Inhalt und Layout ermöglichen und somit zu einem deutlich verständlicheren HTML-Quelltext beitragen.

Wenn auf Tabellen als Gestaltungsmittel nicht verzichtet werden soll oder kann, so sollten im Hinblick auf deren barrierefreie Verwendung die in [Utac] gegebenen folgenden Empfehlungen unbedingt beachtet werden:

1. keine Verwendung des HTML-Header Tags (`<td>`); anderenfalls kann ein Benutzer eines Screenreaders den Eindruck erhalten, er befinde sich in einer Daten-Tabelle, obwohl dies nicht der Fall ist.
2. keine absoluten Größenangaben in Tabellen verwenden, sondern stattdessen proportionale; die Begründung hierfür entspricht derjenigen, die bereits in Abschnitt 2.5.1 vorweggenommen wurde.
3. die Tabellenstruktur so einfach wie möglich halten; mit Hilfe von geschachtelten Tabellen, verbundenen Zellen (dies kann erreicht werden durch die Attribute `colspan` bzw. `rowspan`) und einer Rahmendicke von 0 (d.h. ohne sichtbaren Rahmen), lassen sich nahezu beliebige Layouts realisieren, deren komplexe Struktur dem normalen Besucher der Webseite verborgen bleibt. Die durch Screenreader vorgenommene Linearisierung dieser Seiten kann jedoch leicht auf eine nicht mehr nachvollziehbare textuelle Repräsentierung führen.

2.6.4 Einsatz von JavaScript

Reine HTML-Seiten haben einen ausschließlich statischen Charakter, das heisst, sie sind während ihrer Darstellung im Client (Browser) unveränderbar. Änderungen am Inhalt lassen sich erst durch ein erneutes Laden der Seite (*page reload*) darstellen. Interaktionspotenzial bieten reine HTML-Seiten daher nicht.

Ein anderes Extrem bilden die sogenannten *Rich Internet Applications* (RIAs), welche im weiteren Verlauf dieser Arbeit im Mittelpunkt der Betrachtungen stehen werden. RIAs sind in besonders hohem Maße interaktionsfähig und führen sehr viel Logik clientseitig.

Reine HTML-Seiten und RIAs sind somit zwei Extrema. Als Abstufung "zwischendrin" haben sich Technologien etabliert, die HTML-Seiten um "ein wenig" Dynamik anreichern und sie partiell oder komplett aktualisieren können, ohne dass der Benutzer dies jeweils veranlassen muss. Diese Technologien bedienen sich in der Regel der Skriptsprache "JavaScript" (siehe [SUN]) oder ihrer Ergänzungen bzw. Derivate. Die folgenden Abschnitte beschreiben diese Technologien jeweils im Hinblick auf ihre Barrierefreiheit.

2.6.4.1 JavaScript

JavaScript ist eine Skriptsprache, mit deren Hilfe Funktionen zu Webseiten hinzugefügt werden können, die mit HTML nicht realisierbar wären. Der JavaScript-Quellcode wird entweder direkt in das HTML-Dokument eingebettet (durch die Verwendung des HTML-Tags `<script>`) oder aus einer separaten Datei geladen. In beiden Fällen interpretiert der Browser den JavaScript-Code[8] und führt ihn aus. Auf diese Weise kann der rein statische Charakter von HTML-Seiten aufgehoben und einerseits mit Interaktivität sowie andererseits auch mit visuellen Effekten angereichert werden.

Die Verwendung von JavaScript-Code kann sich jedoch nachteilig auf die Barrierefreiheit auswirken. Beispielsweise könnte eine Webseite von JavaScript derart Gebrauch machen, dass zum Funktionieren des Scripts die Verwendung der Maus Voraussetzung ist. In diesem Fall ist nicht berücksichtigt, dass der Benutzer eventuell auf die Verwendung der Tastatur angewiesen ist, weil er eine Maus nicht verwenden kann. JavaScript kann ereignisbasiert eingesetzt werden, und deshalb macht es Gebrauch von sogenannten *Event Handlern*. Diese reagieren in einer vom Programmierer zu spezifizierenden Weise auf das Eintreten eines bestimmten Ereignisses (*Event*). Beispiele für Ereignisse sind das Anklicken (nebst Loslassen) eines Buttons mit der Maus oder das Drücken einer Taste auf der Tastatur. Wenn sich (um beispielsweise ein Auswahlmenü zu realisieren) die Ereignisbehandlung eines JavaScript-Programms auf von der Maus ausgelöste Ereignisse beschränkt, so ist dieses Menü für Benutzer der Tastatur nicht zugänglich. Umgekehrt kann durch die einseitige Fokussierung (nur) auf Tastaturereignisse jener Benutzerkreis ausgeschlossen werden, dem die Tastatur als Eingabemedium nicht zur Verfügung steht. Allgemein gilt: durch den Einsatz von Event-Handlern, die sich nur auf ein bestimmtes Eingabemedium beziehen, ist eine HTML-Seite also nicht mehr unabhängig vom Eingabegerät verwendbar und damit nicht mehr barrierefrei. Um dies zu vermeiden, sollte entweder ganz auf eine derartige JavaScript-basierte Navigation verzichtet werden oder zumindest stets ein zusammengehöriges Paar von Event-Handlern für Tastatur und Maus zum Einsatz kommen. Das Ereignis `onBlur` ist beispielsweise das Tastatur-Pendant zum Maus-Ereignis `onMouseOut`, ebenso bilden `onFocus` und `onMouseOver` ein entsprechendes Paar.

Die einseitige Fokussierung auf ein Eingabegerät ist aber nicht die einzige Gefahr, beim Einsatz von JavaScript die Barrierefreiheit zu verletzen. Durch JavaScript kann auch Einfluss auf das Browserverhalten genommen werden. Letzteres kann beispielsweise dazu verwendet werden, (neue) Browserfenster zu öffnen (hier sind insbesondere sog. *PopUps* zu nennen) oder offene Fenster zu schließen. Ebenso kann die Funktion der Browser-Buttons "vor" und "zurück" manipuliert werden und infolgedessen durch unerwartetes Verhalten überraschen. Oftmals empfinden selbst nichtbehinderte Benutzer des Internets derartig manipuliertes Browserverhalten als Ärgernis oder zumindest als "lästig". Zu einem ernsthaften Hindernis werden sie aber insbesondere für Screenreader-Benutzer, die auf ein verlässliches Browser-Verhalten in besonderer Weise angewiesen sind.

[8] sofern der Browser dazu in der Lage ist und JavaScript nicht deaktiviert wurde

Desweiteren bietet JavaScript die Möglichkeit, den Inhalt einer Seite (oder Teile davon) dynamisch zu verändern, indem sie beispielsweise in regelmässigen Abständen automatisch – also ohne dass der Benutzer dies veranlasst hat – aktualisiert wird. Während dies für normale Benutzer keinen Nachteil bedeutet, können Screenreader diese regelmäßigen Aktualisierungen nicht darstellen bzw. dem Benutzer nicht vermitteln. Auf die Probleme, die dies verursachen kann, geht der nachfolgende Abschnitt über AJAX näher ein.

Hinsichtlich der Barrierefreiheit von JavaScript nennt [Utac] folgende Voraussetzungen (eigene, sinngemäße Übersetzung):

> "Eine Webseite, die JavaScript-Technologie verwendet, gilt als barrierefrei, wenn sie
>
> - unabhängig vom Eingabegerät (Maus/Tastatur) bedient werden kann,
> - ihren Inhalt auch für alternative Ausgabegeräte zugänglich bereit hält,
> - das Browserverhalten nicht in einer ungewohnten Form manipuliert und
> - eine barrierefreie Alternative für den Fall vorhält, dass die JavaScript-basierte Version nicht barrierefrei gestaltet werden kann."

Der letztgenannte Punkt ist auf zwei Arten realisierbar: einerseits kann eine eigene HTML-Seite, die gänzlich auf JavaScript verzichtet, angeboten werden. Andererseits steht das HTML-Element `<noscript>` zur Verfügung, welches die Möglichkeit bietet, Alternativtext anzugeben, der angezeigt wird, wenn der verwendete Client nicht JavaScript-fähig ist (oder JavaScript deaktiviert wurde). Eine Seite, die dieses Element verwendet, wird dadurch aber nicht automatisch barrierefrei. Denn wenn JavaScript im Browser *nicht* deaktiviert wurde (was auch bei Screenreadern durchaus gewollt sein kann – Gründe hierfür folgen), wird der gegebene Alternativtext gar nicht erst zur Anzeige gebracht. An späterer Stelle in dieser Arbeit werden die Entwicklungsparadigmen *Graceful Degration* und *Progressive Enhancement* vorgestellt (vgl. Abschnitt 5.4.1 auf Seite 129). Die an einem dieser Paradigmen orientierte Entwicklung von Webseiten führt im Ergebnis ebenfalls zur Lösung des hier beschriebenen Problems.

Bei der Entwicklung von Webseiten, die mit JavaScript angereichert werden und gleichzeitig barrierefrei bleiben sollen, muss daher die Entscheidung getroffen werden, ob

- JavaScript nur in dem Umfang verwendet wird, wie der zuvor genannten Definition (noch) Genüge getan werden kann, ob
- durch den Einsatz des HTML-Elements `<noscript>` alternative Darstellungsformen für Clients, die nicht JavaScript-fähig sind, unmittelbar in die jeweilige Seite integriert werden oder ob
- darüber hinausgehender Gebrauch von JavaScript gemacht wird, dafür aber eine alternative Seite zur Verfügung steht, die gänzlich auf JavaScript verzichtet und damit vollständig barrierefrei ist.

Sofern die dritte Alternative Umsetzung findet, wird in der Regel die JavaScript-freie Seite nur mit den wesentlichen Informationen ihres JavaScript-haltigen Pendants aufwarten und sich insoweit in Layout und/oder Funktionalität ein wenig schlichter präsentieren. Im Vorgriff auf eine Definition, die im weiteren Verlauf dieser Arbeit im Abschnitt 5.1.2.1 gegeben wird, kann diese Alternativ-Darstellung als "HTML-Fallback-Version" (vgl. Seite 108) bezeichnet werden.

2.6.4.2 AJAX

Im Bemühen, den statischen Charakter von HTML-Seiten um Dynamik zu ergänzen, hat sich in den vergangenen Jahren der Begriff "AJAX" (ein Akronym für *Asynchronous JavaScript And XML*, siehe [Gar05] für eine detailliertere Einführung) etabliert, welcher verschiedene Technologien zusammenfasst. Wie bereits aus dem Namen ersichtlich ist, stellt JavaScript einen wesentlichen Bestandteil dieses Technologien-Konglomerats dar, und deshalb ist auch bei der Verwendung von AJAX die Frage zu beanworten, inwieweit dies die Barrierefreiheit einer Webseite einschränkt.

Zum einen folgt aus der Tatsache, dass JavaScript ein elementarer Bestandteil der AJAX-Technologie ist, unmittelbar, dass die im vorstehenden Abschnitt beschriebenen Problemfelder auch für die Verwendung von AJAX gelten. Zum anderen ist es aber gerade der Kerngedanke von AJAX, der sich im Hinblick auf Barrierefreiheit negativ auswirkt:

Im traditionellen Ansatz (ohne AJAX) kontaktiert ein Webclient (der Browser) einen Webserver, dieser liefert eine komplette Seite an den Browser zurück, welcher diese Seite dann zur Anzeige bringt. Wesentlich ist hier, dass stets die gesamte Seite vom Server (erneut) an den Client übertragen wird, und zwar auch dann, wenn sich nur ein kleiner Bereich verändert hat. Mit Hilfe von AJAX ist es hingegen möglich, tatsächlich nur einen Teil einer kompletten Webseite zu aktualisieren, und das insbesondere ohne dass dies seitens des Benutzers ausgelöst wurde. Triviale Anwendungsbeispiele sind die Anzeige einer Uhrzeit oder auch die Darstellung eines aktuellen Aktienkurs-Verlaufs. Zur Realisierung dieser partiellen Aktualisierung wird das DOM[9]-Objekt verwendet, welches dynamische Änderungen an Inhalt, Struktur und Layout der Seite ermöglicht, sodass sich mit Hilfe von JavaScript und dem DOM-Objekt einzelne Elemente der Seite unmittelbar ansprechen und manipulieren lassen. Somit lässt sich auf einfache Art und Weise zum Beispiel der Text eines konkreten Absatzes austauschen.

Ein weiteres Einsatzszenario für AJAX liegt aber auch in der Verarbeitung von Formularen: bereits während der Eingabe durch den Benutzer können z.B. Plausibilitätskontrollen durchgeführt oder Teilergebnisse berechnet (und sogleich auch dargestellt) werden. Somit können auf AJAX-basierende Webanwendungen bereits zu den *Rich Internet Applications* gezählt werden, die in Kapitel 4 eine detailliertere Einführung erfahren werden.

Unabhängig von der Komplexität einer auf AJAX basierenden Webanwendung bleibt die Kernidee stets gleich: im Hintergrund (d.h. ohne explizites Auslösen durch den Benutzer) wird ein Server kontaktiert, und nachdem dessen Antwort empfangen wurde, wird im Browser ein Teil der angezeigten Seite durch die empfangene Antwort ersetzt oder in einer anderen Form darauf reagiert. (Im Falle einer Falscheingabe auf einem Formular könnte dies beispielsweise die Ausgabe einer Fehlermeldung sein.) Eine Gegenüberstellung der Funktionsweise von traditionellen und AJAX-basierten Webanwendungen skizziert Abbildung 2.3 (eigene Interpretation und Erweiterung nach [Gar05]). Sie macht deutlich, dass bei synchronen Webanwendungen entweder der Client oder der Server eine Phase der Inaktivität durchläuft. Während dieser Phase wartet der Client (der Server) auf den Abschluss einer Aktion, welche die Gegenseite (Server bzw. Client) momentan ausführt. In asynchronen Webanwendungen können Client und Server hingegen parallel arbeiten und werden nicht durch das Warten auf den Abschluss einer Aktivität der Gegenseite ausgebremst. Dies impliziert zugleich, dass ein Kontakt zum Server nicht stets durch den Benutzer initiiert werden muss, sondern auch selbstständig durch die AJAX-Engine ausgelöst werden kann.

[9] DOM = Document Object Model

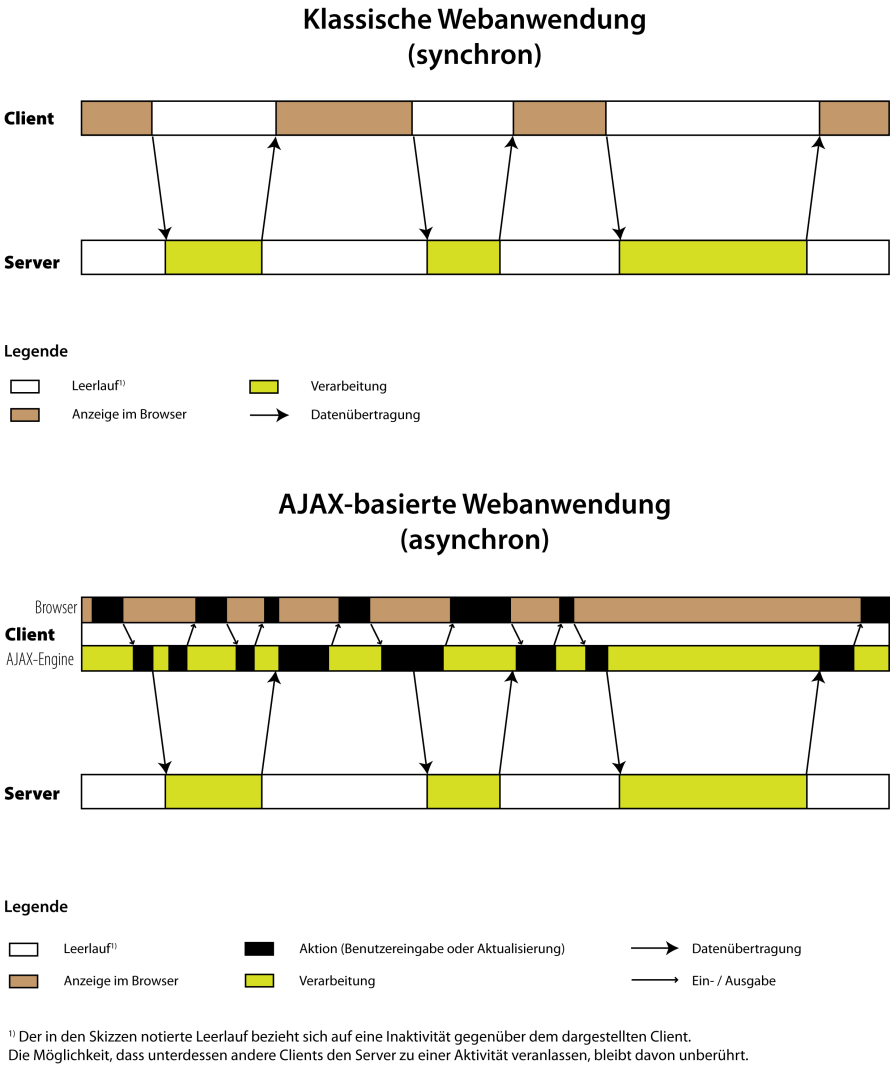


Abbildung 2.3: Gegenüberstellung der Funktionsweise traditioneller und AJAX-basierter Webanwendungen (nach [Gar05])

Diese im Hinblick auf *Usability* (siehe Seite 88) durchaus vorteilhafte Idee kann sich allerdings auch als nachteilig herausstellen, nämlich dann, wenn hinterfragt wird, ob die partielle(n) Änderung(en) auf der Seite auch wirklich vom Benutzer wahrgenommen werden können. Einerseits betrifft diese Frage wiederum blinde Menschen (d.h. Screenreader-Benutzer), andererseits darüber hinaus durchaus aber auch normalsichtige Benutzer. Je nach Größe und Art der Darstellung kann es nahezu jedem Menschen entgehen, dass durch AJAX ein Teil der Seite soeben aktualisiert wurde. Ein Vorschlag, wie diesem Problem begegnet werden kann, präsentiert Lindermann in "The Yellow Fade Technique"[Lin04]. Die Kernidee ist, die jüngsten Aktualisierungen automatisch für kurze Zeit farblich hervorgehoben darzustellen (konkret beschreibt [Lin04] in diesem Zusammenhang einen gelben Hintergrund, der innerhalb eines Zeitintervalls kontinuierlich verblasst; dies ist auch zugleich namensgebend für den Mechanismus) und ihnen dadurch zur Beachtung zu verhelfen. Diese Technik wird auch in [Kra05] empfohlen. Weil sie aber ihrerseits wieder rein visueller Natur ist, kann sie nicht für Screenreader verwendet werden. Deshalb schlägt [Kra05] zudem vor, immer dann, wenn sich der Inhalt (oder ein Teil davon) verändert hat, ein akustisches Signal auszulösen.

Die Praxistauglichkeit und die Tragfähigkeit dieser Vorschläge sind jedoch in Frage zu stellen, denn

- je nachdem, wie häufig regelmässige Aktualisierungen vorkommen, können die akustischen Hinweise den Benutzer eines Screenreaders eher enervieren als ihm helfen,
- es ist unklar, wie die akustischen Signale gegeben werden sollen, wenn sich mehrere Inhaltsbereiche zugleich ändern. (Soll nur *ein* Signal ausgelöst werden? Oder ist es besser, stattdessen *mehrere* auszulösen?)

Akustische Signale können zwar eine Hilfe sein, sie können aber auch gegenteilig wirken. Letztlich sollte es deshalb dem Benutzer selbst überlassen werden, ob er diese Signale wünscht oder nicht, indem er sie ein- oder abschalten kann.

2.6.4.3 Empfehlungen für die Benutzung von AJAX

Die vorstehende Diskussion macht klar, dass JavaScript und darauf aufbauende weiterführende Ansätze im Hinblick auf Barrierefreiheit mit Vorsicht zu verwenden sind. Technische und persönliche (gesundheitliche) Barrieren können die Verwendung dieser Technologien erschweren oder gar unmöglich machen. Für den normalen Benutzer hingegen sind die Möglichkeiten von JavaScript, AJAX und weiterführenden Ansätzen durchaus zu begrüßen, und sie finden deshalb auch eine sehr rasche Verbreitung im World Wide Web. Als Quintessenz ergeben sich die folgenden Empfehlungen für die Verwendung von AJAX im Hinblick auf Barrierefreiheit.

- Der Benutzer ist darauf hinzuweisen, dass JavaScript bzw. AJAX zur Anwendung kommen.
- Der Benutzer soll durch ein Signal (akustisch und/oder optisch) darüber informiert werden, dass eine Änderung des Inhalts vorgenommen wurde; aber:
- Diese Benachrichtigungsfunktion soll optional, d.h. benutzerseitig ein- oder ausschaltbar sein.

Gibson gibt in [Gib06] ebenfalls diese Empfehlungen, ergänzt sie aber noch um die Forderungen, dass

- der Benutzer einen Hinweis erhalten soll, wenn ein Element den Fokus erhält, ohne dass der Benutzer dies veranlasst hat,
- die Navigation auf der Seite besonders Rücksicht auf Benutzer von Screenreadern nehmen soll, indem zum Beispiel Überschriften mit Bedacht platziert werden[10] und
- die Aktualisierung von bestehenden Elementen ("*Updating*") dem Hinzufügen neuer Elemente ("*Creating*") vorzuziehen ist, um ebenfalls die Navigation für Screenreader zu vereinfachen.

[10] Diese dienen Screenreadern als Strukturierungshilfe.

2.6.5 Weitere Prinzipien

Die nachstehende Aufstellung vervollständigt (unter Verzicht auf weitere Erläuterungen) die Liste der von der WebAIM-Initiative genannten Prinzipien:

- klar verständlich und in einfacher Sprache schreiben
- kontrastreiche Farben verwenden, um die Lesbarkeit auch für farbenblinde Menschen zu gewährleisten
- Text in skalierbaren Schriften darstellen (d.h. als "echten Text" schreiben anstatt ihn als Grafiken einzubetten)
- Trennung von Inhalt und Layout durch die Verwendung von Cascading Style Sheets (CSS)

Über diese Empfehlungen hinaus werden noch weitere in einschlägigen Foren und Newsgroups diskutiert. Die hier vorgestellten *best practices* erheben also nicht den Anspruch, vollständig zu sein. Sie sind jedoch insoweit vollständig, als dass sich alleine mit ihnen das Framework zur Erstellung barrierefreier Webseiten (Kapitel 3) und somit im weiteren Verlauf damit auch die barrierefreien HTML-Fallback-Lösungen von RIAs erstellen lassen.

2.7 Barrierefreiheit und multimediale Inhalte

HTML ist von seiner Struktur her nicht geeignet, multimediale Inhalte wiederzugeben. Zwar ist es möglich, animierte Bildformate (z.B. gif-Dateien) darzustellen, die ähnlich einem "Daumenkino" eine Sequenz von mehreren Bildern enthalten, die in kurzer Folge nacheinander angezeigt werden und auf diese Weise den Effekt eines Films simulieren. Diese Animationen wirken jedoch archaisch im Vergleich zu Videoformaten, die eigens für den multimedialen Einsatz im World Wide Web konzipiert worden sind. Ein besonders verbreiteter Vertreter dieser Gattung ist die Shockwave/Flash-Technologie, die von Macromedia (heute: Adobe) entwickelt wurde. Auf diesem Format basiert auch das "Flash Video Format" (flv), das von Adobe ins Leben gerufen wurde. Während flv-Dateien lediglich Videos bzw. Animationen beinhalten, bietet das weitergehende "vollwertige" Flash-Format ("swf"-Dateien) Möglichkeiten zur Interaktion mit dem Benutzer (siehe unten). Beide Formate nutzen den kostenlos verfügbaren Flash-Player als Wiedergabeplattform, und nach Aussagen von Adobe ist dieser auf 97% der weltweiten Computer installiert. Auch mobile Endgeräte verwenden zunehmend Flash (derzeit noch in der reduzierten Variante names *Flash light*, in naher Zukunft aber auch die Vollversion) als technisches Fundament für multimediale Inhalte. Einen enormen Popularitätsgewinn hat das flv-Format insbesondere dadurch erfahren, dass Plattformen wie *YouTube*[11] sowie die Webauftritte großer Nachrichtenagenturen multimediale Inhalte mittlerweile nur noch im flv-Format anbieten.

Sofern es nur um das Abspielen von Audio- und Videosequenzen geht, existieren neben Flash auch noch weitere Multimediaformate, wie zum Beispiel das *Windows Media Format* (wmf) oder das *Real*-Format. Allen diesen Formaten ist gemein, dass sie ihrer Natur nach nicht unmittelbar barrierefrei sein können. Auf Seiten, auf denen derartige Dateien Verwendung finden, müssen deshalb Alternativ-Darstellungen bzw. -beschreibungen eingesetzt werden, damit diese

[11] http://www.youtube.com

Seiten auch von behinderten Benutzern in vollem Umfang inhaltlich erfasst werden können. Audio-Transskriptionen oder Untertitel auf Videos sind in diesem Zusammenhang etablierte Möglichkeiten.

Neben dem Einsatz als reine Multimedia-Plattform zur Wiedergabe von Audio- und Videosequenzen kommt der Flash-Technologie noch eine besondere Bedeutung zu. Im Unterschied zu anderen Formaten bietet diese zusätzlich auch ein weitreichendes Animations- und Interaktionspotenzial. Insbesondere zu Marketing- und Werbezwecken wird deshalb anstelle von traditionellem HTML gerne die Flash-Technologie eingesetzt. Die Grenzen zwischen Animation und Web-Applikation sind in diesem Umfeld nicht klar zu ziehen. Vielmehr beginnt hier das Spektrum der "Rich Internet Applications", das fortan im Mittelpunkt dieser Arbeit stehen wird: Inhalte werden *reichhaltig* präsentiert, d.h. die reine Informationsvermittlung wird angereichert durch eine ansprechende, interaktive Präsentation, die allein mit reinem HTML, CSS und JavaScript nicht möglich wäre.

Mit *Silverlight* entwickelt Microsoft derzeit eine zu Flash ganz ähnliche Plattform, und weitere Entwicklungen dieser Art mögen noch folgen (so verfolgt auch Sun Microsystems mit Java FX einen ähnlichen Ansatz). Momentan kann Flash aber aufgrund dessen technischer Reife und Verbreitung noch eine Monopolstellung zugesprochen werden. Deshalb – und der praktischen Teils dieser Arbeit sehr stark auf der Flash-Plattform basiert – kommt diesem Format nachstehend noch eine eingehendere Betrachtung zu. Im weiteren Verlauf der Arbeit werden die zu Flash im Wettbewerb stehenden Formate noch genannt, eine Beschäftigung mit diesen erfolgt aber nur insoweit, als sie sich voneinander in Bezug auf Barrierefreiheit unterscheiden.

2.7.1 Vor- und Nachteile von Flash in Bezug auf Barrierefreiheit

Der Einsatz der Flash-Plattform für Internet-Auftritte birgt im Hinblick auf Barrierefreiheit Chancen und Risiken, d.h. Vor- und Nachteile. In diesem Abschnitt sollen sie einander gegenüber gestellt werden.

2.7.1.1 Vorteile für die Barrierefreiheit bei der Benutzung von Flash

Grundsätzlich bietet Flash Möglichkeiten, die Zugänglichkeit und Barrierefreiheit von multimedialen Inhalten drastisch zu *erhöhen*. So nennt beispielsweise [Utaa] die folgenden Vorteile von Flash:

- Flash kann Inhalte auf verschiedene Arten aufbereiten und darstellen, beispielsweise rein textuell, grafisch sowie auch akustisch.

- Flash-Inhalte basieren in der Regel auf Vektor-Elementen, die sich verlustfrei beliebig skalieren lassen. Somit kann der Inhalt optischvergrößert werden, ohne dass dies einen Qualitätsverlust mit sich bringt. Dies ist wichtig für Menschen mit Beeinträchtigung des Sehvermögens, weil diese oftmals sogenannte *screen magnifier* einsetzen, welche den Bildschirminhalt stark vergrößern. Pixel- bzw. Bitmap-basierte Formate wie beispielsweise jpg, gif u.ä. erlauben eine verlustfreie Skalierung hingegen nicht.

- Mit Hilfe von Flash können Tastatur- bzw. Mauseingaben weitaus komplexer verarbeitet werden, als dies in HTML möglich ist, beispielsweise dadurch, dass sie in Kombinationen

Einschränkung	Problembeschreibung	Abhilfe
Taubheit	Audioinhalte können nicht wahrgenommen werden.	Transskript zur Verfügung stellen
Epilepsie	Blinkende Inhalte können einen epileptischen Anfall auslösen.	Verzicht auf kritische Frequenzen (zwischen 2 und 55 Hz)
Einschränkungen in der Motorik	Eingaben mit der Maus und/oder der Tastatur können nicht präzise genug getätigt werden.	wurde im Text bereits erläutert
Kognitive Einschränkungen	Das Erfassen von Inhalten dauert länger als bei anderen Menschen.	Verzicht auf Inhalte, die zeitkritisch sind oder Möglichkeit, das Zeitfenster benutzerseitig verlängern zu lassen
Sehbehinderungen	wurde im Text bereits erläutert	analog zu den Empfehlungen im Text.
Blindheit	Tastaturkürzel zur Bedienung der Animation/Applikation überschneiden sich ggf. mit denen zur Steuerung des Screenreaders	Überschneidungen vermeiden

Tabelle 2.1: Mögliche Probleme für Menschen mit Einschränkungen beim Einsatz der Flash-Technologie und deren Lösungen (nach ([Utaa])

mit weiteren Tasten (STRG, ALT u.ä.) getätigt werden[12].

- Die Multimedia-Möglichkeiten von Flash können Menschen mit kognitiven Schwächen helfen, den Inhalt besser zu erfassen (durch optische Hervorhebungen, Grafiken und Animationen, Soundunterstützung, etc.).

- Insbesondere folgt aus der Fähigkeit, Sound auszugeben, dass Flash seine eigenen Inhalte selbst akustisch ausgeben kann und somit den Einsatz von Screenreadern unterstützt oder sogar entbehrlich werden lässt.

Es lassen sich auch Bedienkonzepte wie beispielsweise Drag-'n'-Drop realisieren. Sie können durchaus einen Beitrag zur Barrierefreiheit leisten, weil sie ungeübten oder ggf. kognitiv eingeschränkten Benutzern die Bedienung erleichtern[13]. Gleichzeitig aber sind derartige Bedienkonzepte in Bezug auf Barrierefreiheit auch kontraproduktiv, denn Benutzern, die nicht mit einer Maus arbeiten (können), bleiben sie (sofern keine weiteren Vorkehrungen getroffen werden) unzugänglich.

2.7.1.2 Nachteile für die Barrierefreiheit bei der Benutzung von Flash

Neben diesen Vorteilen, die Flash für die Zugänglichkeit von Internetseiten bieten kann, resultieren aus dem Einsatz dieses Formats aber auch klare Nachteile für Menschen mit bestimmten Einschränkungen bzw. Behinderungen (die in diesem Fall nicht nur auf Sehbehinderungen limitiert sind). Tabelle 2.1 nennt diese Nachteile und zeigt zudem Lösungsansätze auf ([Utaa]).

[12] Dies kann sowohl ein Vorteil als auch ein Nachteil sein. Vorteilhaft ist es, wenn Benutzer, die Probleme im Umgang mit der Maus haben, eine alternative Navigation durch Tastenkombinationen vorfinden. Dies erweist sich jedoch zugleich als Nachteil für diejenige Gruppe von Benutzern, die Probleme bei der Handhabung von Tastaturen bzw. komplexeren Tastenkombinationen haben (vgl. Fußnote zur Arbeit über "Low Bandwidth Input" auf Seite 6).

[13] Dieser Einschätzung liegt die Annahme zugrunde, dass ein Bedienkonzept wie Drag-'n'-Drop eine intuitivere Form der Bedienung darstellt, als sie beispielsweise von eher abstrakten grafischen Elementen typischer Benutzeroberflächen (Listen, Drop-Down-Menüs, etc.) angeboten wird.

Auch wenn Flash grundsätzlich Möglichkeiten anbietet, die in Tabelle 2.1 aufgeführten Empfehlungen umzusetzen und somit barrierefrei eingesetzt werden könnte, sind Flash-Inhalte, die diese Empfehlungen berücksichtigen, in der Praxis momentan noch nicht weit verbreitet. Zwar mögen *einige* der Empfehlungen jeweils umgesetzt sein, aber um als wirklich zugänglich und barrierefrei zu gelten, müssten *alle* der vorstehenden Empfehlungen umgesetzt sein – mitunter sogar auch noch weitere. Oder anders ausgedrückt: Flash-Inhalte sind *nicht* barrierefrei, solange nicht alle Empfehlungen *zugleich* umgesetzt worden sind.

2.7.2 Flash und Screenreader

Flash-Inhalte unterscheiden sich gravierend von HTML-Seiten, denn sie liegen nicht in textueller Form vor und lassen sich deshalb nicht linearisieren. Dies aber ist eine wesentliche Voraussetzung für ihre "Darstellbarkeit" durch Screenreader. Eine intensivere Diskussion über das Zusammenspiel von Flash-Inhalten und Screenreadern wird an späterer Stelle in dieser Arbeit erfolgen (Abschnitt 4.5ff. auf Seite 97ff.). Deshalb sollen die Ausführungen über Flash und Barrierefreiheit an dieser Stelle mit der Empfehlung nach [Utaa] schliessen. Sie besagt, dass die Verwendung von Flash dann nicht den Bemühungen um die Barrierefreiheit zuwiderläuft, wenn entweder

- der Inhalt in einer Form präsentiert wird, die unmittelbar für Screenreader erreichbar ist, oder
- der Inhalt durch Flash selbst in eine akustische Form gebracht werden kann und somit den Einsatz eines Screenreaders unnötig macht oder
- eine (nicht flash-basierte) barrierefreie Alternative des Inhalts zur Verfügung steht.

Letzteres entspricht dem Gedanken der *HTML Fallback Lösung*, der auch bereits in Zusammenhang mit JavaScript erwähnt wurde und im weiteren Verlauf dieser Arbeit weiter verfolgt werden wird (vgl. Seite 108ff).

2.8 Der Zusammenhang von Barrierefreiheit und Suchmaschinenoptimierung

Befolgt man bei der Erstellung von Webseiten die in diesem Kapitel zusammengefassten Empfehlungen und *Best Practices*, so weisen die daraus resultierenden Seiten eine saubere inhaltliche Struktur auf, die zudem frei von Layout-Informationen ist. Wie dargestellt wurde, ist eine saubere Struktur eine unabdingbare Voraussetzung für die Verwendung assistiver Technologien. Darüberhinaus profitieren aber auch Suchmaschinen von den semantisch klar analysierbaren HTML-Seiten. Die für Suchmaschinen das Web durchforstenden Programme ("Webcrawler") sind ebenso wie beispielsweise Screenreader auf eine klar erkennbare Semantik angewiesen, um die gewünschten – aussagefähigen – Informationen zu gewinnen. Der Bereich der Suchmaschinen-Optimierung ("Search Engine Optimization", SEO) befasst sich detailliert mit der Aufbereitung von Webseiten, um diese für Webcrawler analysierbar zu machen. Weiterführende Informationen dazu finden sich unter anderem in [Thu07].

2.9 Zur Messbarkeit von Barrierefreiheit

Eine ausschließlich objektive Aussage darüber, ob eine bestimmte Webseite barrierefrei ist oder nicht, lässt sich nicht treffen, denn in ein solches Urteil gehen stets subjektive Bewertungen ein. Eine Seite, die für den einen barrierefrei ist, mag dies für den anderen noch lange nicht sein – selbst wenn bei der Erstellung der Seite alle vorstehenden Regeln und Empfehlungen befolgt wurden. Zu unterschiedlich sind die persönlichen Vorlieben und individuellen Bedürfnisse der Menschen, als dass es *die* barrierefreie Webseite geben könnte. Deshalb ist es sinnvoller, nicht zu fragen, *ob* eine Webseite barrierefrei ist, sondern *wie* barrierefrei sie ist. Um diese Frage zu beantworten, sind Kriterien erforderlich, mit Hilfe derer sich die Barrierefreiheit von Webseiten bemessen lässt. Entsprechende Ansätze sind unter anderem in [TLM09] und [MCG09] beschrieben.

2.10 Ein erstes Fazit zur Barrierefreiheit im traditionellen Web-Umfeld

Moderne Webseiten und Webapplikationen lassen sich auch für Menschen gestalten, die mit alternativen Ein-/Ausgabegeräten arbeiten müssen oder aus sonstigen Gründen die normale Darstellung nicht nutzen können. Insbesondere stellen aktuelle Technologien wie AJAX keinen Widerspruch zur Barrierefreiheit dar, solange ihre Verwendung auch stets potenzielle Barrieren für bestimmte Benutzer(gruppen) im Blick behält. Das bedeutet jedoch, dass bereits bei der Erstellung der Webseiten eine Sensibilität für Barrierefreiheit und vor allem das entsprechende Problembewusstsein vorhanden sein muss.

Im Zeitalter des “Web 2.0” ist das Web zu einem “Mitmach-Web” geworden, das alle Internet-Nutzer einlädt, eigene Inhalte beizutragen. Die dazu eigentlich notwendigen Kenntnisse über die Gestaltung von Webseiten werden gekapselt durch Tools, die sich vordergründig wie ein Textverarbeitungsprogramm bedienen lassen, im Hintergrund aber HTML-Code erzeugen. Somit kommt diesen Tools die Aufgabe zu, barrierefreien Inhalt zu produzieren. Ein Überblick über aktuelle Tools dieser Art und die Konzeption und Implementierung eines eigenen diesbezüglichen Ansatzes sind Gegenstand des folgenden Kapitels.

Da die Verwendung von AJAX bereits als Einstieg in die Sphäre der “reichhaltigen” Web-Anwendungen angesehen werden kann, ergibt sich aus der vorausgegangenen Diskussion bereits ein Leitfaden für die Entwicklung von barrierefreien *Rich Internet Applications* (RIAs), wobei dazu noch keine neuen Technologien oder Frameworks verwendet werden müssen, sondern nur das entsprechende Problembewusstsein sowie eine daraus resultierende sorgfältige Gestaltung erforderlich sind.

Derartige RIAs sind aber nur ein kleiner Teil der gesamten RIA-Welt. Der weitaus größere Teil der aktuellen RIAs basiert derzeit auf der Flash-Technologie. Vergleichbare Technologien von Wettbewerbern (z.B. Microsofts “Silverlight” und JavaFX) befinden sich noch in frühen Entwicklungsphasen, werden aber vermutlich schon bald weitere Verbreitung finden und mit Flash vergleichbar sein. Auch ist davon auszugehen, dass noch weitere RIA-Technologien folgen werden. Eines ist ihnen jedoch in jedem Fall gemein: sie erfordern neue Konzepte, um mit ihnen eine barrierefreie Darstellung des Inhalts zu gewährleisten. In Kapitel 4 wird das Thema Barrierefreiheit deshalb nochmals aufgegriffen und in Bezug auf seine Bedeutung und Umsetzbarkeit in RIAs diskutiert.

Kapitel 3

Barrierefreie Authoring Tools

"Meaning can be as important as usability in the design of technology"[ST09]

3.1 Motivation

Die im vorstehenden Kapitel zusammengestellten *Best Practices* und Empfehlungen für eine barrierefreie Gestaltung von Webseiten stellen bei jeweils individueller Betrachtung keine neuen Entwicklungen dar. Auch in ihrer dort präsentierten Gesamtheit geben sie nur das wieder, was sich bereits als geübte Praxis und *de facto*-Standards etabliert hat.

In der Praxis kann jedoch beobachtet werden, dass längst nicht alle Empfehlungen stets eine konsequente Umsetzung erfahren. Über die Gründe dafür kann an dieser Stelle keine Aussage getroffen werden, jedoch liegt die Vermutung nahe, dass oftmals mangelndes Problembewusstsein und/oder unzureichende Kenntnisse im korrekten Umgang mit HTML und CSS die Ursachen sind. Auch der Vorwurf, dass sich Nutzerfreundlichkeit und Barrierefreiheit häufig den Layout- und Designvorstellungen der Webdesigner unterordnen müssen, wird in einschlägigen Foren immer wieder diskutiert.

Gleichzeitig stellt sich die Frage nach Lösungen für dieses Problem. Einerseits können diese Lösungen – ganz dem Gedanken des "Web 2.0" folgend – auf die freiwillige Mitwirkung der Nutzergemeinde (hier konkret: auf die Benutzer ohne Sehbehinderungen) setzen. Andererseits aber zeigt die Diskussion in den Foren, dass eine softwareseitige Unterstützung von Barrierefreiheit bei der Erstellung von Webinhalten zunehmend erwartet wird.

Einen Ansatz der erstgenannten Kategorie verfolgt IBM mit dem "Social Accessibility Project"[1]:

3.1.0.1 Das Social Accessibility Project von IBM

Dieses Projekt wurde von IBM im August 2008 ins Leben gerufen. Es lädt sehbehinderte Benutzer ein, Seiten, die sie mit Hilfe eines Screenreaders nicht korrekt erfassen können, in eine Datenbank aufzunehmen. Andere Nutzer können dann diese Datenbank abfragen, die entsprechenden Seiten im wörtlichen Sinne an*sehen* und die nicht zugänglichen Informationen

[1] `http://services.alphaworks.ibm.com/socialaccessibility/`

in der Datenbank hinterlegen (d.h. darin also die unzugänglichen Informationen ergänzen). Somit entsteht mit der Zeit eine Datenbank, die als Ersatzquelle für diejenigen Informationen verwendet werden kann, die auf den originären Webseiten (für sehbehinderte Menschen) nicht zugänglich sind.

Abgesehen davon, dass dieses Projekt den Gedanken des "Web 2.0", das Internet zu einem "Mitmach-Netz" auszuformen, in dem jeder Benutzer mitwirken kann, auf eine sehr nützliche Art weiterführt, geht von dieser Initiative aber auch eine Bestätigung für die vorliegende Arbeit und die im folgenden entwickelten Ideen aus: auch im Jahr 2008 existierten – obwohl die Empfehlungen zur barrierefreien Gestaltung von Webseiten bereits einige Jahre alt waren – noch derart viele Seiten im World Wide Web, welche diesen Empfehlungen *nicht* folgten und (für Sehbehinderte) unzugänglich blieben, dass ein Branchen-Schwergewicht wie IBM den Bedarf sah, mit einem Projekt zu helfen. Und auch aktuell – am Ende des Jahres 2009 – lebt dieses Projekt noch fort.

In diesem Zusammenhang ist das Ergebnis einer im April 2009 in Deutschland durchgeführten Umfrage des Marktforschungsinstituts TNS EMNID von Interesse. Diese Umfrage hat ergeben, dass "80% der [deutschen] Internetnutzer überhaupt keine Kenntnis darüber haben, was barrierefreie Webseiten sind, und 60% der [deutschen] Internetnutzer nicht wissen, dass u.a. Menschen mit Behinderungen und ältere Menschen Webseiten nicht nutzen können, wenn sie nicht barrierefrei programmiert sind."[TNS09]

Das Ergebnis dieser Umfrage ist sicherlich nicht auf die Berufsgruppe der Web-Designer und Web-Programmierer übertragbar. Jedoch lädt das "Web 2.0" ja gerade *alle* Internetnutzer dazu ein, eigene Inhalte zu publizieren, und zahlreiche Tools und Editoren ermöglichen auch in HTML unerfahrenen Internet-Benutzern das Erstellen von Webseiten. Wenn das Problembewusstsein zur Barrierefreiheit derart unterentwickelt ist, wie es durch die Umfrage konstatiert wird, dann ist es umso mehr erforderlich, dass die Tools und Editoren zur allgemeinen Verwendung "im Hintergrund" daran mitwirken, dass barrierefreie Webseiten entstehen. Das in diesem Kapitel vorgestellte WebTool und das sich daraus ergebende Framework verfolgen genau dieses Ziel.

3.1.0.2 Barrierefreiheit in Authoring Tools

Der Fokus dieser Arbeit soll fortan auf Techniken und Software-Lösungen liegen, die bereits bei der *Erstellung* von Seiten helfen, diese barrierefrei zu halten. Denn Webseiten entstehen heutzutage nicht mehr zwangsläufig unmittelbar durch das manuelle Erstellen bzw. Editieren einer HTML-Datei in einem einfachen Editor. Stattdessen existiert eine große Zahl von Softwarelösungen am Markt, die jeweils damit werben, Webseiten auch ohne Kenntnisse von HTML und CSS einfach generieren zu lassen. Zu unterscheiden sind an dieser Stelle Programme, welche die *Gestaltung* einer Webseite in den Mittelpunkt stellen sowie jene, bei denen der *Inhalt* der Seiten die zentrale Rolle spielt. Trotz dieser unschiedlichen Ausrichtungen, ist beiden Arten von Programmen gemein, dass sie dem Benutzer eine WYSIWYG[2]-Oberfläche zur Verfügung stellen, mit Hilfe derer er seine Webseite(n) gestaltet bzw. mit Inhalt versieht. Je nach Ausrichtung der Software spricht sie insbesondere solche Benutzer an, die keine tieferen Kenntnisse – mitunter auch gar keine – in HTML und CSS besitzen. Bezüglich der HTML-Qualität ihrer Seiten müssen sich diese Benutzer auf die Versprechungen der Software verlassen.

[2]WYSIWYG: "What-you-see-is-what-you-get" – eine gängige Beschreibung für Programme, in denen die Darstellung von Dokumenten am Bildschirm der späteren Ausgabe am Endgerät entspricht

Aus technischer Sicht kann im Zusammenhang mit HTML gar kein WYSIWYG-Ansatz realisiert werden, denn die Verantwortung für die Darstellung obliegt dem darstellenden System (Browser) des aufrufenden Benutzers und ist von zahlreichen Faktoren bestimmt (unter anderem: Bildschirmgröße und Bildschirmauflösung sowie anderen individuellen Einstellungen). Dennoch versuchen die Hersteller der WYSIWYG-Editoren, mit Hilfe allerlei "Tricks" eine systemübergreifend konsistente Darstellung der mit ihrer Software erstellten Seiten zu erreichen. Allzuoft wird eine solche Funktionalität mit invalidem und/oder nicht mehr barrierefreiem Quellcode erkauft.

Dies führt zu der Überlegung, inwieweit man bereits die Werkzeuge zur Erstellung von Webseiten (sog. *"Authoring Tools"*) dahingehend erweitern kann, dass sie entweder barrierefreie Ergebnisse produzieren oder zumindest warnen, wenn das produzierte Ergebnis nicht als barrierefrei validiert werden kann. Insbesondere im Hinblick auf die bereits genannten einschlägigen Rechtsvorschriften zur Barrierefreiheit muss diese Funktionalität als äußerst wichtiges Leistungsmerkmal derartiger Software angesehen werden.

Als Maßstab für die Unterstützung von Barrierefreiheit können die im vorangegangenen Kapitel erläuterten *Best Practices* herangezogen werden. Einige Applikationen zur Entwicklung von Webseiten stellen bereits eigene Validatoren, mit denen sich die in Entwicklung befindliche Seite immer wieder auf ihre Barrierefreiheit hin überprüfen lässt, zur Verfügung. Ferner gibt es auch die Möglichkeit, die Seiten online (beim W3C) auf Standardkonformität und Barrierefreiheit überprüfen zu lassen[3]. Letztlich ist es aber "nur" das: ein Überprüfen. Stellt sich nämlich dabei heraus, dass das Design die Anforderungen an Barrierefreiheit nicht erfüllt, ist ein – mitunter aufwändiges – Redesign erforderlich. Berücksichtigt man zudem, dass manche Webseiten von verschiedenen Leuten zugleich gepflegt (was den disziplinierten Umgang mit Barrierefreiheit ggf. erschwert) und dass professionelle Webseiten-Tools mit ihren Validatoren bei weitem nicht überall dort, wo Webseiten produziert werden, zum Einsatz kommen, wird der Bedarf an softwareseitiger Unterstützung erkennbar, um bereits bei der Erstellung der Seiten bzw. deren Inhalt eine barrierefreie Ausgabe sicherzustellen.

Das vorliegende Kapitel beschäftigt sich mit dem soeben motivierten Problembereich. Zunächst gibt es einen Überblick über den aktuellen Stand der Technik im Umfeld der *Content-Management-Systeme* (CMS). Den unbestreitbaren Vorteilen dieser Systeme stehen jedoch verschiedene Nachteile gegenüber – diese erfahren im kommenden Abschnitt eine detaillierte Erläuterung.

Im weiteren Verlauf wird gezeigt, dass der Einsatz aufwändiger und teils auch kompliziert zu administrierender Content-Management-Systeme oftmals gar nicht notwendig ist, sondern durch einfache "Bordmittel" des Betriebssystems sowie kleiner Hilfsprogramme ersetzt werden kann. Am Beispiel einer auf der Programmiersprache *Perl* basierenden Implementierung wird vorgeführt, wie eine komplette Website[4] barrierefrei konzipiert, unterhalten und durch mehrere Autoren inhaltlich gepflegt werden kann, ohne dass sich diese Autoren im einzelnen um das Thema Barrierefreiheit und Standardkonformität kümmern oder ein Content-Management-System einsetzen müssen.

Das einführende Beispiel dient zugleich als Ideengeber für ein Framework, das fortan im Mittelpunkt des praktischen Teils dieser Arbeit stehen wird. Für sich alleine genommen kann die Perl-basierte Implementierung für kleine bis mittlere Websites bereits als simples Surrogat eines Content-Management-Systems verwendet werden. Im weiteren Verlauf werden die zugrunde liegenden Ideen dieser Implementierung abstrahiert, um somit ein Framework zu gewinnen,

[3] `http://validator.w3.org/` für HTML-Code und `http://jigsaw.w3.org/css-validator/` für CSS-Dateien

[4] Anmerkung zur Begriffsverwendung: mit Web**seite** ist eine einzelne Internet-Seite gemeint, wohingegen Web**site** einen kompletten Internet-Auftritt (bestehend aus mehreren Webseiten) meint.

das flexibel und skalierbar verschiedene Facetten bei der barrierefreien und standardkonformen Erstellung von Webseiten unterstützt. Dieses Framework wird auch für den in einem späteren Kapitel beschriebenen Ansatz, HTML-Fallback-Lösungen von RIAs automatisch generieren zu lassen, als technisches Fundament verwendet.

Mit einem Nachweis, dass das Framework den einschlägigen Richtlinien des W3C-Konsortiums für Authoring Tools genügt, schliesst dieses Kapitel.

3.2 Die Authoring Tool Accessibility Guidelines des W3C

Die Bandbreite verfügbarer Softwarelösungen zur Erstellung und Verwaltung von Webseiten reicht vom simplen HTML-Editor bis hin zu komplexen Content-Management-Systemen. Entsprechend unterschiedlich sind auch die jeweiligen Benutzerkreise dieser Programme sowie die Schwerpunkte, auf welche sie vorrangig ausgerichtet sind. Eines ist ihnen jedoch stets gemein: sie dienen dazu, die redaktionelle Arbeit zu unterstützen, und dies oftmals derart, dass sie seitens der Benutzer keine HTML- und CSS-Kenntnisse verlangen. Programme dieser Art können damit unter dem Begriff "Autorensystem" zusammengefasst werden[5].

Die bereits erwähnte *Web Accessibility Initiative* des W3C hat bereits im Jahr 2000 unter der Bezeichnung *"Authoring Tool Accessibility Guidelines"* (kurz: ATAG) Richtlinien eingeführt, welche von Autorensystemen idealerweise beachtet werden sollten. Aktuell liegen sie in der Version 2.0 von Dezember 2008 vor. Von der Beachtung dieser Richtlinien erhofft sich das W3C, dass "zusätzlich zu ihrem Nutzen für die Zugänglichkeit [...] viele der in der Spezifikation angesprochenen Grundsätze – wie z.B. die Bedeutung der Erstellung und Bewahrung eines gültigen Markups – die Interoperabilität des Webs im Allgemeinen [fördern]"[W3C08a]. Über diesen Aspekt hinaus gehen die Richtlinien noch einen Schritt weiter. Ihr Fokus auf die Barrierefreiheit bzw. Zugänglichkeit ist nicht nur auf die von den Tools erzeugte Ausgabe beschränkt, sondern bezieht sich auch auf die Tools selber. Sind auch diese barrierefrei verwendbar? Sind sie zugänglich? Obwohl diese Richtlinien im Jahr 2000 verabschiedet wurden (und damit also aus der Ära vor dem "Web 2.0" stammen), antizipieren sie durch das Einbeziehen der Autorensysteme selbst in die Richtlinien den Mitmach-Gedanken des heutigen Webs: jeder soll Autor im World Wide Web sein können, und auch Menschen mit Behinderungen müssen in der Lage sein, nicht nur verfügbare Inhalte barrierefrei zu konsumieren, sondern auch zu produzieren.

Mit den Richtlinien werden die folgenden Ziele verfolgt:

- Das Autorensystem soll barrierefrei zu bedienen sein.
- Das Autorensystem soll barrierefreie Ausgaben erzeugen.
- Das Autorensystem soll zum Erstellen barrierefreien Inhalts anregen.

(Die vorgenannten Ziele stellen eine eigene, sinngemäße Übersetzung des Originaltextes dar.)

[5] Der Begriff Autorensysteme ist im Allgemeinen nicht nur auf HTML-Editoren beschränkt, sondern schließt z.B. auch integrierte Programmierumgebungen ("*Integrated Development Environment*", abgekürzt IDE) mit ein. Im Kontext dieser Arbeit sind mit Autorensystemen aber in der Tat redaktionelle Softwarelösungen wie z.B. Content-Management-Systeme gemeint.

3.2.1 Die sieben Richtlinien im Einzelnen und die *Checkpoints*

Die ATAG umfassen im Kern sieben Richtlinien, die von Autorensystemen zu beachten sind. Diese Richtlinien lauten im englischen Original sowie in einer eigenen, sinngemäßen Übersetzung ins Deutsche wie folgt:

1. Support accessible authoring practices.
   (Unterstütze barrierefreie Arbeitsabläufe der Autoren!)
2. Generate standard markup.
   (Erzeuge standardisierte Ausgaben!)
3. Support the creation of accessible content.
   (Unterstütze die Erstellung barrierefreien Inhalts!)
4. Provide ways of checking and correcting inaccessible content.
   (Biete Möglichkeiten zum Überprüfen und Korrigieren von nicht-barrierefreiem Inhalt an!)
5. Integrate accessiblity solutions into the overall "look and feel".
   (Integriere Lösungen zur Barrierefreiheit in das gesamte Erscheinungsbild [der Software]!)
6. Promote accessibility in help and documentation.
   (Mache in den Hilfe-Texten und in der Dokumentation auf Barrierefreiheit aufmerksam!)
7. Ensure that the authoring tool is accessible to author with disabilities.
   (Stelle sicher, dass das Autorensystem auch für Menschen mit Behinderungen zugänglich ist.)

Die sieben Richtlinien formulieren jeweils einen "Wunsch" an die Software, bleiben aber ansonsten recht allgemein. Deshalb wird jede Richtlinie um jeweils mehrere Konkretisierungen (*Checkpoints*) ergänzt. Diese Konkretisierungen präzisieren, wie die in der jeweils übergeordneten Richtlinie ausgedrückte Absicht erreicht werden kann bzw. soll.

3.2.2 Prioritätsstufen und Erfüllung der Ziele

Anhand der zuvor erwähnten Checkpoints lässt sich bemessen, wie richtlinienkonform ein konkretes Tool ist. Jedem Checkpoint ist eine von vier Prioritätsstufen ("*Priority Levels*") zugeordnet. Mit Hilfe dieser Prioritätsstufen wird zum Ausdruck gebracht, wie wichtig der jeweilige Checkpoint für das Erreichen der oben genannten Ziele ist. Dazu werden die Stufen 1 bis 3 (wobei 1 für *essential*, 2 für *important* und 3 für *beneficial* stehen) sowie "relative Priorität" als die in dieser Aufstellung unterste Prioritätsstufe unterschieden.

Konformitätsgrade

Zur Klassifizierung von Autorensystem-Software hinsichtlich der Richtlinienkonformität sehen die Richtlinien drei Konformitätsgrade ("*Conformance Levels*") vor:

Conformance Level "A": Jeder Checkpoint der Priorität 1 sowie alle relativen Prioritäten werden erfüllt.

Conformance Level "Double-A": Jeder Checkpoint der Prioritäten 1 und 2 sowie alle relativen Prioritäten werden erfüllt.

Conformance Level "Triple-A": Jeder Checkpoint der Prioritäten 1, 2 und 3 sowie alle relativen Prioritäten werden erfüllt.

In Bezug auf Barrierefreiheit werden Autorensysteme somit vergleichbar(er) und erlauben, den eigenen Status Quo in einer definierten und nachvollziehbaren Bewertung zu kommunizieren.

Als kleine Anekdote am Rande sei hier bemerkt, dass bereits die Bezeichnungen der drei *Conformance Levels* bewusst barrierefrei gewählt wurden. Das W3C legt Wert darauf, dass anstelle von "AA" die Bezeichnung "Double-A" verwendet wird (analog gilt dies für "AAA", das zu "Triple-A" wird), denn wenn diese Bezeichnungen von einem Screenreader ausgegeben werden, lassen sie sich wesentlich besser verstehen, als wenn "AA" bzw. "AAA" buchstabengetreu vorgelesen würden.

3.2.3 Neuformulierung des Ziels dieses Kapitels

Das eingangs motivierte Ziel dieses Kapitels lässt sich nun wie folgt neu formulieren: es soll ein Framework enstehen, das als Autorensystem verwendet werden kann und dabei einen möglichst hohen *Conformance Level* aufweist. Das wesentliche Kriterium, das dieses Framework von existierenden Lösungen unterscheidet, ist dessen Schlankheit, weil es keine in sich geschlossene Softwarelösung darstellt, sondern lediglich die Generierung und Validierung der Seiten in den Mittelpunkt stellt und dazu konsequent auf standardisierte und etablierte Techniken (dies sind die im vorangegangenen Kapitel erläuterten *Best Practices*) zurückgreift.

Heutzutage kommen vorrangig Content-Management-Systeme als Autorensysteme zur Anwendung. Der Bedarf für das zu entwickelnde Framework ergibt sich unter anderem auch aus den Nachteilen, welche die Verwendung von Content-Management-Systemen mit sich bringen kann. Deshalb soll im folgenden Abschnitt eine kurze Übersicht über diese Systeme gegeben sowie deren Vor- und Nachteile erörtert werden.

3.3 Content-Management-Systeme

Content-Management-Systeme (CMS) haben ihren Ursprung in großen Verlagen bzw. Redaktionen und in einer Zeit, in der das gedruckte Medium noch im Vordergrund stand. Die Redakteure haben ihre Beiträge in derartige Systeme eingestellt und brauchten sich nicht um das konkrete spätere Layout kümmern. Lediglich der Inhalt wurde von ihnen dem System zugeführt, und dies war letztlich auch namensgebend. Einem CMS kommen nach [EGK06] die folgenden Aufgaben zu:

- Darstellung der Inhalte,
- Bereitstellung von Möglichkeiten, die Inhalte zu erstellen und zu bearbeiten,
- Organisation von Inhalten,
- Unterstützung des *Workflows* (beispielsweise: einreichen/erstellen, Korrektur lesen, Korrekturen vornehmen, veröffentlichen, ggf. auch eine Versionskontrolle),

- Zugriffssteuerung (d.h. Unterstützung der Zusammenarbeit großer Autoren-Teams dergestalt, dass Rechte für das Schreiben und Lesen von Dokumenten individuell gesetzt werden können.)

Weder der Begriff "Inhalt" noch das Zielmedium, auf dem die Inhalte ausgegeben werden sollen, erfahren in dieser Aufgabenauflistung eine Konkretisierung. Wenn heutzutage Content-Management-Systeme erwähnt werden, so sind damit in aller Regel solche Systeme gemeint, die Inhalte von Webseiten verwalten und dementsprechend auch elektronische Ausgabeformate als Zielmedium adressieren. Um Missverständnissen vorzubeugen, wird deshalb oftmals der Begriff *Web-Content-Management-Systeme* (WCMS) verwendet. Die Betrachtungen in diesem Abschnitt beziehen sich stets auf derartige WMCS.

3.3.1 Trennung von Inhalt und Layout

Hinter dem vermeintlichen profanen ersten Punkt der obigen Aufzählung der Aufgaben verbirgt sich eine der wesentlichen Stärken von (W)CM-Systemen. Er impliziert, dass die Inhalte unabhängig vom Layout gehalten werden und dadurch eine sehr variable Möglichkeit besteht, die Inhalte für verschiedene Zielformate aufzubereiten. Dies können verschiedene Versionen in HTML/CSS sein (beispielsweise einmal mit besonders kontrastreichen Farben, einmal mit besonders großen Schriften, etc.), aber ebenso lassen sich die Ausgaben für mobile Endgeräte optimieren. Gute WCM-Systeme bieten zudem noch die Option an, die Inhalte als druckbare PDF-Version auszuliefern.

3.3.2 WYSIWYG-Editierfunktion: Vor- und Nachteil zugleich

Als zweite Aufgabe von CM-Systemen wird die Bereitstellung von Möglichkeiten, die Inhalte zu erstellen und zu bearbeiten, genannt. Insbesondere werben die Hersteller dieser Systeme damit, dass für redaktionelle Arbeiten keinerlei HTML-, CSS- oder gar Programmierkenntnisse benötigt werden. Unbestritten kann dies ein immenser Vorteil sein, denn auf diese Weise ist der Kreis der Autoren nicht auf jene mit entsprechendem Wissen beschränkt. Im Hinblick auf Barrierefreiheit stellt sich dies aber zugleich auch als ein gravierender Nachteil heraus: zum Erstellen bzw. Bearbeiten der Inhalte bringen gängige WCMS-Lösungen eigene WYSIWYG-Editoren mit. Diese erlauben – insbesondere den Benutzern ohne Kenntnisse in HTML und CSS – das Erfassen und Editieren ihrer Texte in einer aus gängigen Textverarbeitungsprogrammen in der Regel bereits gewohnten Art und Weise. Das bedeutet, dass anstelle von semantischen Auszeichnungselementen wie z.B. `h1`, `h2`, `p` usw. manuelle Formatierungen (z.B. fette und/oder kursive Schrift) vorgenommen werden, um beispielsweise Überschriften und andere strukturelle Elemente vom übrigen (Fließ)-Text abzuheben. Einerseits ist dies für die Autoren ein klarer Vorteil: sie müssen keinerlei HTML-Kenntnisse haben und sind dennoch in der Lage, Inhalte für das Web zu produzieren. Die Aufbereitung des editierten Inhalts in HTML wird später vom WCMS übernommen.

Im Hinblick auf die Barrierefreiheit des generierten HTML-Codes liegt hier aber andererseits auch die größte Schwachstelle von WCM-Systemen, denn durch ein traditionelles WYSIWYG-Editieren eines Textes besteht die Gefahr, die semantische Struktur des Textes zu verwässern: wenngleich derartige Seiten bei der Anzeige in einem Browser durchaus die gewünschte Erscheinung aufweisen, so wurde hier die strikte Trennung von Layout und Inhalt verletzt. Dies hat zur Folge, dass der Quellcode nicht mehr anhand seiner semantischen Struktur analysiert und somit von Screenreadern und Textbrowsern nicht mehr optimal wiedergegeben werden kann

(vgl. zur Trennung von Inhalt und Layout die Ausführungen in Kapitel 2). Bedauerlicherweise wird auf diesen Problembereich in der Dokumentation vieler CM-Systeme nicht hingewiesen. Im Gegenteil: in Zusammenhang mit CM-Systemen wird teilweise in der "Selbstvermarktung" (Eigenwerbung), aber auch in unabhängigen Artikeln in verschiedenen Medien immer wieder darauf hingewiesen, dass Inhalte auch gänzlich ohne HTML-Kenntnisse erstellt werden können. Derartige Aussagen konterkarieren das in der sechsten ATAG-Richtlinie (vgl. Seite 33) zum Ausdruck kommende Bemühen, auch in der Dokumentation dem Thema Barrierefreiheit Raum zu geben und die Benutzer der CM-Systeme für dieses Thema zu sensibilisieren.

3.3.3 Statische versus dynamische WCM-Systeme

Zum Zeitpunkt der Erstellung dieser Arbeit existiert eine große Auswahl sowohl an kommerziellen als auch an frei verfügbaren WCMS-Lösungen am Markt. Die Entscheidung für oder gegen bestimmte Lösungen ist in der Regel eine Frage der individuellen Präferenzen und mithin subjektiv. Unabhängig vom konkreten Produkt ist die folgende Klassifizierung interessant:

Statische WCMS: In statischen Systemen werden die Inhalte so abgelegt, dass sie unmittelbar für die Darstellung zur Verfügung stehen. Das bedeutet, dass vollständige ("fertige") HTML-Seiten auf einem Web-Server vorgehalten werden. Aus Sicht des Benutzers werden zwar auch hier Layout und Inhalt voneinander getrennt, jedoch werden die Seiten nach Abschluss einer Bearbeitung sofort zur Darstellung (automatisch) aufbereitet und in dieser Form dann gespeichert. Der Vorteil dieses Ansatzes ist, dass die Seiten auch von Suchmaschinen gefunden werden können, weil sie bereits auf einem Server existieren.

Dynamische WCMS: Dynamische Systeme generieren eine HTML-Seite erst beim Abruf durch einen Client. Auf dem Server liegt der reine Inhalt, und seine Aufbereitung zur Ausgabe erfolgt serverseitig nach Anforderung. Dem Vorteil, auf diese Weise stets aktuelle Seiten auszuliefern[6], stehen als Nachteile einerseits eine verzögerte Reaktionszeit (Dauer, bis die Seite vom Server erstellt wurde) sowie andererseits insbesondere eine Nicht-Verfügbarkeit für Suchmaschinen gegenüber.

Die Unterscheidung zwischen statischen und dynamischen Systemen wird zu einem späteren Zeitpunkt in einem anderen Zusammenhang erneut für diese Arbeit von Bedeutung sein.

3.3.4 Barrierefreiheit und CMS

Gemäß [Hei04b] können CM-Systeme als barrierefrei angesehen werden, wenn sie die folgenden Anforderungen erfüllen:

- Ausgabe von strukturiertem Code mit Überschriften, Absätzen und anderen Elementen
- Automatische Überprüfung der Seite auf Validität vor der Veröffentlichung
- Unterstützung bei der Erstellung barrierefreier Inhalte (Sprachwechsel, Abkürzungen usw.)

[6] Beispielsweise können so Navigationsstrukturen, Werbeeinblendungen und weitere Elemente noch aktuell in die Seite integriert werden.

- Benutzerdialoge zum Einbinden von Medien (Alternativtexte für Bilder und ähnliches)
- Barrierefreier Zugang zur Administration und zu allen Funktionen
- Saubere Trennung von Struktur, Layout und Inhalt über Templates
- Verwendung von Cascading Stylesheets zur Formatierung der Darstellung
- Vermeidung von Layout-Tabellen und Frames

Da CMS-Software in die Kategorie der Autoren-Systeme eingeordnet werden kann, lassen sich die aktuell verfügbaren CM-Systeme – wie weiter oben beschrieben – hinsichtlich ihrer Barrierefreiheit anhand der ATAG-Konformitätsstufen (vgl. Seite 33) bewerten. Ein Vergleich der aktuell am Markt verfügbaren CM-Systeme[7] soll jedoch nicht Gegenstand dieser Arbeit sein. Denn ein solcher Vergleich könnte aufgrund der Vielzahl von Systemen, ihrer (teilweise) recht unterschiedlichen Ausrichtungen, Schwerpunkte und technischen Fundamente sowie aufgrund des regelmäßigen Erscheinens neuer Versionen oder gar Systeme ohnehin nur eine vage Momentaufnahme darstellen und bereits nach kurzer Zeit veraltet und nicht mehr korrekt sein. Vermutlich aus demselben Grund sind zum Zeitpunkt der Erstellung dieser Arbeit auch online keine aktuellen Vergleiche verfügbar. Der aktuellste war bereits 2 Jahre alt[8]. Die Recherchen, die für diesen Abschnitt über CMS angestellt wurden, haben jedoch zu den nachstehenden Fragen geführt. Sie sollen als Entscheidungshilfe bei der Auswahl für (oder auch gegen) ein bestimmtes CMS verwendet werden können und insoweit die zuvor genannten Anforderungen sowie die ATAG-Richtlinien ergänzen.

1. Bietet das CMS Design-Vorlagen, die im Hinblick auf eine Verwendung in Screenreadern optimiert sind?
2. Erzeugt das CMS Navigationsstrukturen, die auch durch die Verwendung der Tastatur zu bedienen sind (vgl. dazu auch Abschnitt 2.6.1 auf Seite 12 sowie Abschnitt 4.4 auf Seite 90)?
3. Verzichtet das CMS auf Layouttabellen?
4. Wird Barrierefreiheit nativ unterstützt? Oder müssen zunächst weitere Module (*Add-Ons*) installiert werden?
5. Wird dem Thema Barrierefreiheit in der Dokumentation des CMS genügend Raum gewährt? Werden die im CMS enthaltenen diesbezüglichen Funktionalitäten hinreichend erläutert? Oder müssen diese Informationen aus externen Quellen – wie beispielsweise Benutzerforen – gewonnen werden?

3.3.5 Ein Fazit zur Verwendung von Content-Management-Systemen

Generell sind Content-Management-Systeme ein geeignetes Mittel, um einen Webauftritt barrierefrei zu unterhalten. Durch die Verwendung von Layout-Vorlagen und die strikte Trennung von Layout und Inhalt bieten sie die Möglichkeit, Probleme mit der Barrierefreiheit durch *eine* zentrale Änderung an der Layout-Vorlage zu beheben. Diese Änderung wirkt sich unmittelbar

[7] Eine aktuelle und nach verschiedenen Kriterien sortierte Übersicht gibt zum Beispiel die Webseite `http://en.wikipedia.org/wiki/List_of_Content_Management_Systems`.

[8] `http://juicystudio.com/article/choosing-an-accessible-cms.php`

auf den gesamten Web-Auftritt aus. Diesem Vorteil steht jedoch der Nachteil gegenüber, dass oftmals keine strukturierten HTML-Ausgaben produziert werden, sondern stattdessen durch Formatangaben im Quellcode die gewünschte Struktur mittels optischer Hervorhebungen "simuliert" wird. Überschriften *sehen aus* wie Überschriften, sind aus semantischer Sicht jedoch keine mehr. Das zentrale Problem von Content-Management-Systemen fasst Mike Davis, ein in Barrierefreiheit erfahrener Web-Entwickler, der aktuell für Yahoo arbeitet, sehr prägnant wie folgt zusammen:

> "*It's great that the writing process is workflowed, but the content itself is treated as nothing but a blob of text.*" [Dav04a]

Unabhängig von Fragen der Barrierefreiheit ist der Einsatz von Content-Management-Systemen nicht immer notwendig oder angemessen. Für kleine bis mittlere Web-Auftritte kann der Umstieg auf CM-Systeme durchaus unwirtschaftlich sein, denn der Einsatz von Content-Management-Systemen löst in der Regel einen recht hohen Installations-, Konfigurations- und Einarbeitungsaufwand aus. Auch hier ist einer der Vorteile dieser Systeme – die feingranular mögliche Vergabe von Berechtigungen – zugleich auch ein potenzieller Nachteil, weil dies den administrativen Aufwand – und damit die Kosten – maßgeblich beeinflusst. Auch Folgekosten für die Schulung von Anwendern müssen – selbst bei einer Entscheidung für kostenlose Open-Source-CM-Systeme – berücksichtigt werden. Hinzu kommt, dass mit dem Einsatz von CM-Systemen auch ein Stück administrative Freiheit aufgegeben wird, denn die Verwaltung von Dateien kann nicht mehr nach freien Wünschen, sondern nur noch in den vom jeweils eingesetzten System gesteckten Grenzen erfolgen.

Aus diesem Grunde kann nicht eine grundsätzliche Empfehlung für den Umstieg auf Content-Management-Systeme gegeben werden. Stattdessen stellt sich die Frage, ob es nicht möglich ist, mit einfachen "Bordmitteln" des Betriebssystems sowie kleiner zusätzlicher Tools die Vorteile, die CM-Systeme bieten, nachzubilden, ohne dabei jedoch auch ihre Nachteile in Kauf nehmen zu müssen. Genau dieser Frage geht der verbleibende Teil dieses Kapitels nun nach.

3.4 Das WebTool-Konzept und dessen prototypische Implementierung

Betrachtet man die Vorteile, welche Content-Management-Systeme für sich reklamieren, so stellt sich die Frage, ob es tatsächlich des Einsatzes derartiger Software-Lösungen bedarf, um in den Genuß dieser Vorteile zu gelangen. Ist es nicht sogar möglich, mit Hilfe von vorwiegend "Bordmitteln", also Tools, die bei (Unix-basierten) Betriebssystemen zum Standard-Lieferumfang gehören, ein eigenes, schlankes System aufzubauen, das ebenfalls einen Großteil der Vorteile bereitstellt, dabei aber ohne nennenswerten administrativen Aufwand bei Installation und Betrieb auskommt?

Diese Fragestellung war das auslösende Moment für die Konzeptionierung eines WebTools, welches auf die Entwicklung und Unterhaltung von kleinen und mittelgroßen Webauftritten ausgerichtet ist und die Themen Barrierefreiheit und Einhaltung von Standards in besonderer Weise berücksichtigt.

3.4.1 Begriffsbestimmungen

Einige der in der nachfolgenden Erklärung zum WebTool verwendeten Begriffe sollen vorab präzisiert bzw. definiert werden:

3.4.1.1 Kleine, mittelgroße und große Webauftritte

Die bereits in der vorangegangenen Motivation verwendete Klassifikation "kleine und mittelgroße Webauftritte" bedarf zunächst einer genaueren Spezifizierung. Gemeint ist hier, dass der Webauftritt aus einer Menge von individuellen Seiten besteht, die untereinander verlinkt sind. Im folgenden sollen die Attribute "klein", "mittelgroß" und "groß" wie folgt verwendet werden:

klein: Der Webauftritt besteht aus einer Menge von einzelnen, statischen Seiten, die sich alle auf der identischen Hierarchieebene befinden. Mithin gibt es keine untergeordneten Seiten, sondern nur "gleichberechtigte". Der Inhalt der Seiten ist beliebig, sie werden jedoch unter einem einheitlichen Layout dargestellt.

mittel: Der Webauftritt erfüllt die Definition von "klein", zusätzlich bilden seine Inhalte aber eine hierarchische Struktur. Es gibt also eine erste Navigationsebene ("*Top Level Navigation*"), welche für jeden enthaltenen Eintrag weitere Untereinträge ("*Second Level*", "*Third Level*", usw.) vorhält. Die Hierarchietiefe ist aus technischer Sicht beliebig, jedoch sollten nach einer Empfehlung des W3C Tiefen von mehr als 3 Stufen im Hinblick auf eine für die Besucher noch nachvollziehbare Struktur vermieden werden.

große: Der Webauftritt erfüllt die Definition von "mittel", bietet zudem aber auch dynamische Elemente, wie z.B. eine lokale Suchfunktion oder die Möglichkeit, Kommentare zu hinterlassen.

Wie den vorstehenden Definitionen zu entnehmen ist, spielt die Anzahl der verwalteten Seiten keine Rolle bei der Einteilung in eine der drei genannten Kategorien.

3.4.1.2 Navigationsstrukturen

Mit Navigationsstrukturen sind im Zusammenhang mit dem WebTool diejenigen Elemente auf der Webseite gemeint, die Verweise auf andere Seiten enthalten und *nicht* im eigentlichen Inhaltstext (weißer Bereich in Abbildung 3.1 auf Seite 42) vorkommen. Ob die Seiten, auf die verwiesen wird, unmittelbarer Bestandteil des eigenen Webauftritts sind oder extern liegen, ist dabei unerheblich.

Im Allgemeinen zu unterscheiden sind abstrakte von konkreten Navigationsstrukturen. Abstrakte Strukturen enthalten die eigentlichen Informationen über die Navigationshierarchie und können als Datenmodell (*Model*) verstanden werden. Konkrete Navigationsstrukturen hingegen entsprechen einer visuellen Repräsentierung einer abstrakten Struktur (*View*), z.B. in Form einer horizontalen oder vertikalen Darstellung (vgl. hierzu auch [Hei04a].) Für das Verständnis der nachfolgenden Ausführungen kann diese Unterscheidung zwischen *Model* und *View* jedoch unbeachtet bleiben.

3.4.1.3 HTML-Inhaltsfragment

Wie aus Abbildung 3.1 hervorgeht, setzt sich eine Seite aus mehreren Teilbereichen zusammen. Im konkreten Beispiel sind dies Titel-, Navigations- und Inhaltsbereich. Der Teil des HTML-Codes, welcher sich (nur) auf den unmittelbaren Inhaltsbereich bezieht, wird im folgenden als *HTML-Inhaltsfragment* bezeichnet.

3.4.2 Anforderungsanalyse

Der Konzeption des WebTools ging eine Aufnahme der Erwartungen und Anforderungen an ein solches System voraus. Die für diese Arbeit wesentlichen Anforderungen, welche die Analyse ergeben hat, können dabei wie folgt kategorisiert und gewichtet werden:

1. notwendige Anforderungen an den Umgang mit Inhalten
   (a) Abstraktion von der Markup-Sprache (HTML): es soll möglich sein, beliebige Auszeichnungssprachen (auch eigene!) zu verwenden
   (b) Unterstützung von Validierungen sowohl für Inhalte, die
      - manuell erstellt und gepflegt werden (z.B. in HTML oder einer anderen Auszeichnungssprache) als auch
      - für Inhalte, die sich aus Daten bzw. Datenquellen generieren lassen (z.B. XML-Daten).
   (c) automatische Erzeugung von Navigationsstrukturen
   (d) Unterstützung von Mehrsprachigkeit[9]
   (e) Mehrbenutzer-Funktion/Unterstützung eines Berechtigungssystems
2. notwendige Anforderungen an das Layout
   (a) strikte Trennung von Layout und Inhalt
   (b) vorlagenbasiertes Layout (Änderungen an der zentralen Vorlage sollen Auswirkungen auf den gesamten Webauftritt haben)
   (c) das Layout muss auf CSS basieren (kein Tabellen-Layout!)
   (d) einheitliches Seitendesign über den gesamten Auftritt hinweg
3. wünschenswerte Anforderungen
   (a) möglichst hoher Grad an Konformität der ATAG-Richtlinien (vgl. *"Conformance Level"* auf Seite 33)
   (b) Bereitstellung von Vorlagen ("*Templates*"), sooft dies sinnvoll möglich ist, insbesondere auch für Daten (in Form von DTD/XML-Schemata und XSLT-Dateien; vgl. Abschnitt 3.4.5 auf Seite 53)
   (c) Verwendung von Dateien, die im gesamten Dateisystem liegen, d.h. keine Beschränkung auf spezifische (Unter-)Ordner
   (d) möglichst geringer/kein Installationsaufwand

[9] Auch dies ist ein Beitrag zur Barrierefreiheit, da auch Sprachen Barrieren bedeuten können.

Mit der Maßgabe, eine Lösung für kleine und mittlere Webauftritte zu finden, ist auch implizit eine Aussage hinsichtlich des darin zu verwaltenden Inhalts verbunden: es wird sich um statische Seiten ohne interaktive Elemente handeln. Folglich wäre die angestrebte Lösung *nicht* geeignet, z.B. einen Webauftritt eines Online-Shops (mit Bestellfunktion) oder ähnliche interaktive Seiten zu gestalten. Vielmehr adressieren sie Webauftritte, die aus statischen (d.h. nicht bei Abruf erzeugten, "dynamischen") Inhalten bestehen. Die Unterscheidung zwischen statischen und dynamischen Web-Auftritten wird im weiteren Verlauf dieser Arbeit im Zusammenhang mit *Rich Internet Applications* nochmals von Bedeutung sein (vgl. Abschnitt 5.1.1 auf Seite 106).

Der in statischen Webauftritten darzustellende Inhalt kann stets einer der folgenden Kategorien zugeordnet werden:

freier Inhalt: Der Inhalt besteht aus freiem, von einem Autor manuell erstellten Text.

datenzentrierter Inhalt: Der Inhalt ergibt sich aus einer Aufbereitung der aus einer Datenquelle bezogenen, ihrer Struktur nach im voraus bekannten Daten.

Ein Beispiel für Inhalte, die der zweiten Kategorie zugeordnet werden können, ist ein Veranstaltungskalender: jede Veranstaltung hat einen Termin, einen Ort und ein Thema. Somit weisen alle Seiten, die Veranstaltungen erläutern, eine stets gleiche Struktur auf. Lediglich der konkrete Inhalt – nämlich eine konkrete Veranstaltung bzw. eine Liste von Veranstaltungen – wird von Seite zu Seite variiert.

Ein besonderes Augenmerk soll bei der Entwicklung der Lösung auf der Validierung *aller* Inhalte liegen. Dies kommt auf Seite 40 in Anforderung 1b zum Ausdruck. Während freie Inhalte z.B. durch die bereits erwähnten Validatoren des W3C auf ihre Standard-Konformität überprüft werden können, würde eine Validierung der datenzentrierten Seiten an der falschen Stelle ansetzen. Wie im folgenden gezeigt wird, ist eine Überprüfung der eigentlichen *Daten*, also nicht der anzuzeigenden HTML-Seite, bereits ausreichend, um der gestellten Anforderung gerecht zu werden.

Die prototypische Implementierung eines solch spezifierten Systems wurde in Form des WebTools vorgenommen. Dieses dient seit Sommer 2007 am Institut für Angewandte Informationsverarbeitung (IAI) der Universität Ulm als Werkzeug für die Unterhaltung des Institutsauftritts im Internet (vgl. dazu Abschnitt 3.4.4). Unabhängig von der konkreten Implementierung soll zunächst aber ein technisches Konzept für ein solches WebTool präsentiert werden.

3.4.3 Das technische Konzept des WebTools

Betrachtet man den beabsichtigten Einsatzzweck des WebTools, so lässt sich daraus zugleich eine Idee zur strukturellen Gestaltung der Seitenvorlage gewinnen. Ein Beispiel für eine derartige Gestaltung zeigt Abbildung 3.1.

Erläuterungen zu Abbildung 3.1:

- Jede Seite präsentiert sich in optisch gleicher Form ("*Corporate Design*"), der Seitentitel ist dabei das einzige Element, das sich im Kopf der Seite jeweils ändert.
- Jede Seite enthält (mindestens) eine Navigationsstruktur; diese wird automatisch generiert und erfordert keinerlei manuelle Pflege.

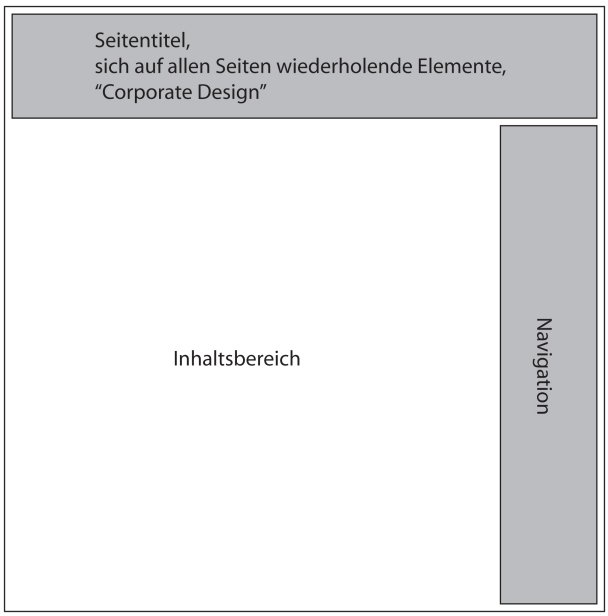


Abbildung 3.1: Grundlegender Aufbau der WebTool-Webseiten

- Der individuelle Seiteninhalt wird im Bereich “Inhalt” dargestellt.
- Den Zusammenbau der einzelnen Elemente erledigt das WebTool automatisch, d.h. lediglich der eigentliche Inhalt bedarf einer manuellen Pflege von Seiten der Benutzer (Redakteure).

3.4.3.1 Der Arbeitsablauf im WebTool

Abbildung 3.2 zeigt, dass sich der gesamte Arbeitsablauf im WebTool in vier Schritte gliedert. Zu Beginn steht die Bereitstellung der Vorlagen. Diese sollte in den Verantwortungsbereich eines in Fragen der Barrierefreiheit und Standardkonformität erfahrenen Web-Designers fallen, denn Voraussetzung für eine wirkungsvolle Funktionsweise des WebTools ist, dass die Vorlagen und Transformationsdateien einen Aufbau haben, der sich an *Best Practices* des vorangegangenen Abschnitts orientiert. Zu den bereitzustellenden Vorlagen zählen die Dokumenttypdefinitionen (bzw. die XML-Schemata), Transformationsdateien (XSLT) sowie die Layout-Vorlage, in welche die generierten Inhalte eingebettet werden. Der zweite Schritt unterscheidet sich je nachdem, ob es sich um inhalts- oder datenzentrierte Seiten handelt. Im Fall von inhaltszentrierten Seiten kann jeder Benutzer (Autor, Redakteur) die Inhalte erstellen. Es werden dazu keine Vorkenntnisse in HTML bzw. barrierefreiem und standardkonformem Design erwartet. Lediglich die inhaltliche Struktur muss in einer geeigneten Form ausgezeichnet werden (z.B. durch eine – auch selbstdefinierte – Syntax). Mit welchen Tools bzw. Editoren der Inhalt erstellt wird, ist dabei unerheblich, woraus sich die Erfüllung der Forderung, selbst barrierefrei (bedienbar) zu sein, ergibt. Die erstellten Seiten werden vom WebTool im dritten Schritt validiert (siehe Abschnitt 3.4.3.6). Sofern die Validierung erfolgreich war, folgt im letzten Schritt die Generierung nebst Veröffentlichung auf dem Webserver. Im Fall von datenzentrierten Seiten durchlaufen diese eine Transformation, liegen danach ebenfalls als HTML-Inhaltsfragmente vor und werden – wie freie Inhaltsseiten auch – in die Layoutvorlagen eingebettet und durch die Navigationsstrukturen ergänzt. Wie Abbildung 3.2 zu entnehmen ist, wird der Benutzer lediglich in einem der vier Schritte aktiv (konkret: in Schritt 2). In der alltäglichen Verwen-

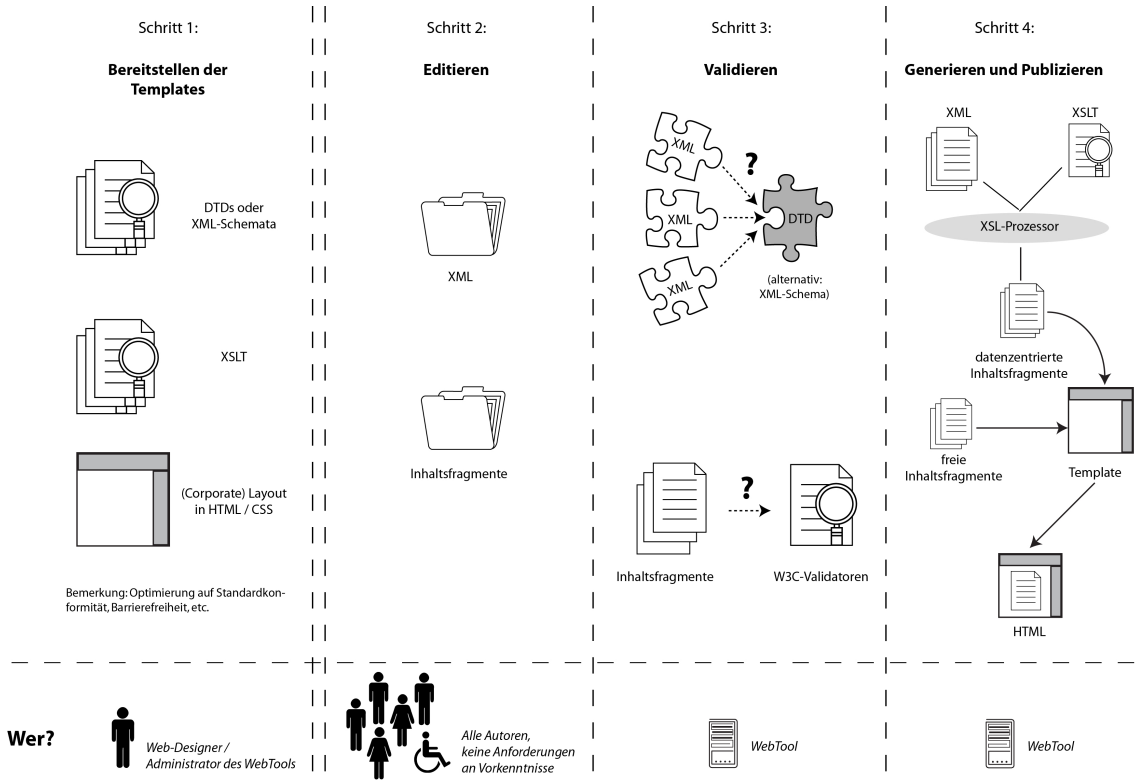


Abbildung 3.2: Die vier grundlegenden Schritte der Funktionsweise des WebTools

dung werden in der Regel wohl nur die Schritte 2 bis 4 durchlaufen, weil es sich bei Schritt 1 um eine vorbereitende Maßnahme handelt, die erst wieder ergriffen werden muss, wenn Änderungen in den Vorlagen gemacht werden sollen. (Dieser Sachverhalt soll durch die doppelte Begrenzungslinie zu Schritt 2 ausgedrückt werden.)

3.4.3.2 Zur Generierung der Seiten

An dieser Stelle scheint ein Widerspruch vorzuliegen: einerseits wurde in der Einführung dieses Abschnitts bemerkt, dass das WebTool für die Erstellung *statischer* Seiten konzipiert wird, andererseits erfolgt aber doch eine *Generierung* von Seiten. Der Widerspruch kann jedoch einfach aufgelöst werden, wenn man betrachtet, zu welchem Zeitpunkt die Generierung der Seiten erfolgt. In dynamischen Systemen erfolgt sie unmittelbar beim Aufruf der Seite, das WebTool hingegen generiert die Seiten dann, wenn die Redaktion Änderungen vorgenommen hat, und stellt das Ergebnis der Generierung auf dem Webserver zur Verfügung. Aus Sicht des Webservers liegen diese Seiten nach der Generierung also in statischer Form vor, obwohl sie selbst das Produkt einer Generierung sind. Hier verhält sich das WebTool analog zu den in Abschnitt 3.3.3 vorgestellten statischen Web-Content-Management-Systemen.

Die Generierung der Seiten erlaubt gleichzeitig auch flexible Änderungen im Layout. Ein einfaches Austauschen der Vorlage (zum Beispiel gegen eine, in der die Navigation am linken statt – wie in Abbildung 3.1 skizziert – am rechten Rand vorhanden ist) wird beim nächsten Generierungslauf das neue Design auf alle Seiten des Webauftritts anwenden.

Die Idee, eine Vorlage (*Template*) zu verwenden, ist für sich alleine betrachtet nicht neu. Einerseits findet sie in Content-Management-Systemen Anwendung, andererseits kennen aber

auch Software-Tools zur Gestaltung von Webseiten wie z.B. das Programm "Dreamweaver" von Adobe diesen Ansatz. Gerade in letzterem muss jedoch jeglicher Inhalt, der in den seitenübergreifenden Bereichen (grau dagestellt in Abbildung 3.1) angezeigt werden soll, bei der Erstellung der Vorlage bereits bekannt sein. Und genau dieser Nachteil an Flexibiltät kann durch die Verwendung des WebTools vermieden werden. Weil quantitative Änderungen an den Inhalten (d.h. Hinzufügen und Entfernen von Seiten) in der Regel auch Änderungen am Aufbau der Navigationsstruktur zur Folge haben, verzichtet das WebTool auf eine manuelle Pflege dieser Strukturen. Stattdessen erzeugt es sie selbstständig bei jedem Generierungslauf.

3.4.3.3 Abbildung der Hierarchie und Navigationsstrukturen

Das erklärte Ziel bei der Entwicklung des WebTools ist es, dieses so einfach und für alle Benutzer so nachvollziehbar wie möglich zu machen. Weil die Navigationsstrukturen automatisch generiert werden, besteht eine erste Herausforderung darin, wie eine Abbildung der Navigationsstrukturen innerhalb des Tools nachvollziehbar realisiert werden kann. Einerseits sollen die (insbesondere auch ungeübten) Benutzer die Strukturen einfach verstehen können, andererseits müssen sie sich aber auch für eine automatische Verarbeitung durch das WebTool eignen. Ferner sollen sie prinzipiell sowohl eine beliebige Hierarchie-Tiefe als auch eine beliebige Hierarchie-Breite unterstützen (wenngleich auch das W3C von Navigationstiefen mit mehr als drei Ebenen abrät).

Ein diesen Anforderungen unmittelbar gerecht werdender Lösungsansatz ist die Abbildung der Hierarchie im Dateisystem. Innerhalb eines ausgezeichneten Ordners liegt die gesamte inhaltliche Struktur in der folgenden Art und Weise vor:

- (Unter-)Verzeichnisse innerhalb einer Ebene eröffnen eine neue Hierarchieebene.
- Reguläre Dateien auf einer Ebene werden mittelbar oder unmittelbar zur Erzeugung der Inhalte (auf dieser aktuellen Ebene) benötigt.

Der zweite Punkt bedarf einer näheren Erläuterung: bei den regulären Dateien kann es sich entweder um jene Dateien handeln, welche die textuellen Inhalte der Seiten enthalten und später in dem in Abbildung 3.1 weiß dargestellten Bereich platziert werden (hier erfolgt eine Verwendung *unmittelbar*), oder um Dateien, welche durch diese Inhalte verwendet (in der Regel referenziert) werden, wie z.B. Bilder, Ton-/Videodateien oder PDF-Dokumente (in diesem Fall liegt eine *mittelbare* Verwendung vor).

Auch an dieser Stelle sei der Hinweis angebracht, dass die Repräsentation eines Webauftritts als (Unter-)Ordnerstruktur kein neuartiger Ansatz, sondern geübte Praxis ist. Was den Ansatz des WebTools jedoch auszeichnet ist, dass die sich daraus ergebende Navigationsstruktur nicht an einer zentralen Stelle (manuell oder, wie im Falle von CM-Systemen, toolunterstützt) vorgehalten und gepflegt wird. Stattdessen wird sie rekursiv durchlaufen und mit jedem Durchlauf neu generiert. Dies hat den Vorteil, dass Änderungen an den Dateien unmittelbar beim nächsten Generierungslauf erfasst werden.

Das WebTool kann nun – beginnend vom obersten Ordner aus – rekursiv diese Struktur durchlaufen und dabei einerseits die Navigationsstrukturen aufbauen sowie andererseits insbesondere auch die Generierung der Seiten (d.h. die Einbettung der Inhalte sowie der Navigationselemente in die Layout-Vorlage) vornehmen. Die dabei erstellten Dateien werden vom WebTool in ein explizit angegebenes Zielverzeichnis (in der Regel wird dies auf dem Webserver liegen) kopiert.

Zur Steuerung der Generierung erlaubt das WebTool die im folgenden beschriebene Konfiguration.

3.4.3.4 Konfiguration der Generierung

Aus der Erklärung zu regulären Dateien im vorstehenden Unterabschnitt lässt sich die folgende Unterscheidung gewinnen: es kann auf jeder Hierarchieebene reguläre Dateien geben, die

- vom WebTool bei der Generierung berücksichtigt werden müssen (um die Inhalte in den Rahmen aus Abbildung 3.1 einzubetten) oder die
- lediglich in ein Zielverzeichnis kopiert werden müssen (d.h. sie sind nicht Gegenstand der Generierung selbst).

Das WebTool bezieht lediglich HTML-Dateien (also Dateien mit der Endung `html` oder `htm`) in die Generierung mit ein. Alle anderen regulären Dateien werden übersprungen. Dennoch kann es gewünscht sein, dass bestimmte HTML-Dateien ebenfalls übersprungen werden sollen. Ebenso würde der Generierungsprozess rekursiv auch alle Unterverzeichnisse durchlaufen. Es kann jedoch durchaus sein, dass die Berücksichtigung bestimmter Unterverzeichnisse mit dem Ziel der Generierung seiner Inhalte nicht gewünscht ist, sondern dass diese Verzeichnisse lediglich so, wie sie sind, in das Zielverzeichnis kopiert werden sollen. Aus diesen beiden Fällen lässt sich bereits die Notwendigkeit erkennen, Konfigurationsoptionen für jede Hierarchie-Ebene angeben zu können. Ein Beispiel dafür, wie eine solche Konfiguration einfach und effizient gestaltet werden kann, wird im Zusammenhang mit der prototypischen Implementierung des WebTools (Abschnitt 3.4.4.3 auf Seite 50) gezeigt.

3.4.3.5 Anforderungen an die Kenntnisse der Benutzer

Dem Beispiel der Content-Management-Systeme folgend, werden für die Verwendung des WebTools keinerlei Kenntnisse der Benutzer hinsichtlich HTML und CSS vorausgesetzt. Im Gegenteil: eine wesentliche Motivation für dessen Entwicklung bzw. Einsatz ist ja gerade, dass es auch in Web-Technologien unerfahrene Autoren in die Lage versetzen soll, standardkonforme, valide und vor allem barrierefreie Ausgaben zu produzieren. Der Administrator des WebTools muß eine geeignete Vorlage bereitstellen und diese dem Tool überlassen. Die konkreten Inhalte werden dann derart in die Vorlage generiert, dass die beschriebenen Anforderungen an den entstandenen HTML-Code erfüllt werden.

Sofern es die Generierung von datenzentrierten Seiten betrifft, ist ihre Qualität durch die Validierung und Transformation (beschrieben in Unterabschnitt 3.4.5) sichergestellt. Autoren, die Inhalte von datenzentrierten Seiten pflegen, müssen deshalb nur – mit einem Editor ihrer Wahl – die entsprechenden XML-Dokumente editieren bzw. erstellen. Eingabefehler, welche die Struktur der XML-Dokumente betreffen, werden durch die spätere Validierung gegen eine DTD- oder eine Schema-Datei erkannt und lösen eine entsprechende Benachrichtigung aus. Somit stellt das Editieren von datenzentrierten Seiten keinerlei Ansprüche an die XML-Kenntnisse des Benutzers. Unberührt von diesen Betrachtungen bleibt natürlich die Frage, ob datenzentrierte Seiten überhaupt einer manuellen Bearbeitung bedürfen, denn XML-Dokumente lassen sich in der Regel auch automatisiert erhalten, beispielsweise aus einer Datenbank. Somit ist es ebenfalls möglich, dass die Inhalte von datenzentrierten Seiten gänzlich

ohne benutzerseitige Eingriffe von der Datenquelle bis hin zur Darstellung als Webseite verarbeitet werden.

Für redaktionelle Arbeiten an freien Inhalten bieten sich zwei Möglichkeiten an: erfahrenen Benutzern mit HTML-Kenntnissen können freie Inhalte unmittelbar in HTML editieren. Der auf diese Weise entstehende Quellcode wird in den Hauptinhaltsbereich der Seite (vgl. weißer Bereich in Abbildung 3.1) eingebettet. Benutzern ohne die entsprechenden HTML-Kenntnisse steht aber auch die Möglichkeit offen, ohne HTML-Kenntnisse Inhalte zu editieren.

Die Idee an dieser Stelle besteht darin, dass sich ein Redaktionsteam auf eine eigene Auszeichnungssyntax für die Struktur der Inhalte verständigen kann. Das WebTool lässt sich dann flexibel um die Unterstützung jeder Syntax dadurch erweitern, dass ein entsprechender Parser als Modul verwendet wird.

Beispiel: Ein Redaktionsteam kann sich darauf verständigen, dass Überschriften stets mit (mindestens) einem Unterstrich beginnen. Je mehr führende Unterstriche eine Zeile besitzt, desto untergeordneter ist die Überschrift in der Dokumenthierarchie. Eine Hauptüberschrift würde demnach genau einen Unterstrich besitzen, eine Überschrift dritter Ordnung entsprechend drei. Nicht durch führende Unterstriche gekennzeichnete Zeilen werden als normaler Fließtext behandelt.

Durch die Einhaltung derartiger syntaktischer Vorgaben, lässt sich ein Dokument auch ohne HTML-Kenntnisse gut strukturieren. Durch die Bereitstellung eines einfachen Parser-Moduls, das sich um die Umsetzung derart ausgezeichneter Dokumente in valides HTML kümmert, kann eine Transformation nach HTML leicht erreicht werden.

Der Vorteil an dieser Stelle – und mithin auch ein wesentliches Alleinstellungsmerkmal gegenüber Content-Management-Systemen – ist, dass hier *keine* Gefahr besteht, dass Benutzer Formatierungen als Ersatz für Strukturelemente verwenden. Sie gestalten das Dokument weiterhin rein strukturell, ohne Verwendung von Layout-Informationen, wodurch die in Anforderung 2a geforderte strikte Trennung von Inhalt und Layout aufrecht erhalten bleiben kann. Der Einsatz einer eigenen Auszeichnungssprache stellt im Unterschied zur Verwendung von HTML eine für ungeübte Benutzer wesentliche Vereinfachung dar, weil

- die Syntax in enger Absprache mit den Benutzern – und damit orientiert an deren Kenntnissen, Fähigkeiten und Erwartungen – entwickelt werden kann (das WebTool erlaubt durch das Hinzunehmen entsprechender Module die Unterstützung jedweder Syntax), und weil
- die XML-/HTML-typische Notation (öffnende/schließende Tags) vor jedem Element entfallen kann. (Für ungeübte Benutzer ist dies sicherlich eine Fehlerquelle.)

3.4.3.6 Validierungen

Eines der wichtigsten Ziele, die mit der Entwicklung des WebTools verfolgt werden, ist, das Erstellen von standardkonformen Seiten so einfach wie möglich zu machen und Eingabefehler, die eine Verletzung des Standards zur Folge haben würden, bereits sehr frühzeitig zu erkennen. Die weiter oben bereits getroffene Unterscheidung zwischen “freiem” und “datenzentriertem” Inhalt greift erneut bei der Implementierung der Validierung:

Validierung freier Inhalte: Sofern beim Bearbeiten freier Inhalte eine eigene Auszeichnungssyntax verwendet wurde, besteht – Fehlerfreiheit im Transformator-Modul vorausgesetzt – keine Gefahr, dass die generierten Seiten ungültig werden. Jedoch ist nicht auszuschliessen, dass den Inhalten doch Formatierungsinformationen mitgegeben wurden (in Form einer *"HTML-Injection"*). Diese können vor der Generierung durch einen entsprechenden Parser herausgefiltert werden[10]. Die ebenfalls mögliche Vorgehensweise, unmittelbar HTML-Fragmente zu erstellen, welche dann bei der Generierung in die Vorlage eingebettet werden, ist im Hinblick auf die Validierung nicht optimal. Das HTML-Inhaltsfragment ist – wie dies in der Definition auf Seite 40 auch zum Ausdruck kommt – lediglich ein Fragment und kein vollständiges (X)HTML-Dokument. Mithin würden alle Validierungsversuche aus genau diesem Grund fehlschlagen. Abhilfe kann hier durch einen speziell angepassten Parser geschaffen werden, der sich auf die Untersuchung der vorhandenen Auszeichnungselemente beschränkt. Eine weitere Möglichkeit ist die Validierung der fertig generierten (und mithin vollständigen) Seite und nicht nur des HTML-Inhaltfragments. Dies widerspricht jedoch dem eigentlichen Gedanken des WebTool, vor oder spätestens bei der Generierung ungültige Inhalte auszusondern.

Validierung datenzentrierter Inhalte: Eine ausgesprochen flexible und universell einsetzbare, zugleich aber auch einfach zu verwendende Möglichkeit, Daten strukturiert abzulegen, bietet die Verwendung der *Extensible Markup Language* (XML). Die einfache Verwendbarkeit folgt aus der Tatsache, dass es sich bei XML-Dateien um reine Text-Dateien handelt, die mit jedem beliebigen Texteditor bearbeitet werden können. Gleichzeitig existieren aber auch speziell auf XML-Dateien ausgerichtete Editoren, welche für in XML ungeübte Benutzer eine weitere Erleichterung darstellen.

Ausschlaggebend für die Verwendung von XML zur Verwaltung datenzentrierter Inhalte im WebTool ist jedoch ein anderes Kriterium: XML-Dateien lassen sich einerseits validieren (entweder gegen eine Dokumenttypdefinition [DTD] oder gegen ein Schema), andererseits aber auch in andere Formate mit Hilfe von Stylesheets transformieren. Validierung und Transformation zusammen dienen als Garant dafür, dass gültige XML-Daten nach der Generierung durch das WebTool stets verlässlich in einer vorhersehbaren und standardkonformen HTML-Repräsentierung vorliegen werden. Ungültige XML-Daten hingegen werden bereits *vor* der Generierung erkannt und aus der Menge der zu generierenden Seiten ausgeschlossen. Eine präzisere Erläuterung der Validierung und Transformation folgt in Abschnitt 3.4.5.

Unabhängig davon, ob später datenzentrierte oder freie Inhalte eingebettet werden, muss andererseits auch die Vorlage (*Template*) valides (X)HTML sein. Diese Vorlage ist jedoch nur einmalig zu erstellen, kann danach auf ihre Validität hin überprüft werden und steht fortan zur Verfügung, ohne dass sie einer erneuten regelmäßigen Validierung bedarf. Der Leitgedanke des WebTools geht davon aus, dass zur Realisierung der Vorlagen die in Kapitel 2 vorgestellten *Best Practices* zur Anwendung kommen.

3.4.3.7 Mehrsprachigkeit

Die auf Seite 40 gezeigte Liste der Anforderungen an das WebTool nennt unter Punkt 1d die Unterstützung von Mehrsprachigkeit. Diese kann ebenfalls einen Beitrag zur Barrierefreiheit leisten, nicht nur dadurch, dass tatsächlich unterschiedliche Sprachen (deutsch, englisch,

[10] Einen derartigen Ansatz hat die britische Rundfunkanstalt BBC vor einigen Jahren mit ihrem Tool *Betsie* (`http://betsie.sourceforge.net/`) verfolgt. Ziel der BBC war, HTML-Seiten von jeglichen Formatierungsinformationen zu befreien, um somit den reinen Inhalt der Seite zu extrahieren. Dieser sollte dann einem Screenreader als textuelle Version des Inhalts zum Vorlesen übergeben werden.

usw.) bereitgestellt werden, sondern auch dadurch, dass innerhalb einer Sprache verschiedene Sprachstile existieren. Für Menschen mit kognitiven Schwächen kann beispielsweise eine vereinfachte Sprachversion vorgehalten werden.

Zur Unterscheidung verschiedener Sprachversionen dient dem WebTool – wie den meisten Webservern ebenfalls – ein Appendix am Dateinamen. So steht beispielsweise `index.html.de` für die deutschsprachige Version der Index-Seite und `index.html.en` für die englischsprachige. Welche Seite beim Besucher der Seiten zur Anzeige gebracht wird, ergibt sich aus den Spracheinstellungen dessen Browsers.

Bei der Generierung nimmt das Webtool aufgrund des Appendix die entsprechenden sprachlichen Unterscheidungen vor:

- Die Navigationsstruktur wird in derjenigen Sprache in die Ausgabedatei hineingeneriert, in der die Quelldatei vorliegt. Die Informationen darüber, wie die Einträge in die Navigationsstrukturen je nach Sprache lauten sollen, können aus den Konfigurationsinformationen gewonnen werden.
- Wenn eine Seite auch in anderen Sprachen vorliegt, so generiert das Webtool die entsprechenden Links (symbolisiert durch Nationalflaggen) in die Seite hinein. Somit wird der Besucher der Seite auf das Vorliegen unterschiedlicher Sprachversionen aufmerksam gemacht.
- Bei der Generierung von datenzentrierten Seiten können ebenfalls verschiedene Sprachversionen erstellt werden. Dies kann auf zwei unterschiedliche Arten realisiert werden. Eine technische Erläuterung dazu erfolgt in Abschnitt 3.4.5.

Seiten, welche keinen Sprachappendix aufweisen, werden bei der Generierung von der soeben beschriebenen Unterscheidung ausgenommen. Im fertig generierten Webauftritt werden sie somit *immer* angezeigt – unabhängig von der Spracheinstellung des Browsers. Somit *kann* eine Seite in verschiedenen Sprachen vorliegen, sie *muss* aber nicht. Dies gewährt dem Benutzer des WebTool für jede Seite die größtmögliche Flexibilität. Die Unterstützung von Mehrsprachigkeit wird im weiteren Verlauf dieser Arbeit im Zusammenhang mit *Rich Internet Applications* noch mehrfach aufgegriffen.

3.4.4 Prototypische Implementierung des WebTools

Der Internet-Auftritt des Instituts für Angewandte Informationsverarbeitung basiert seit Sommer 2007 auf einer prototypischen Implementierung des WebTools. Die wesentlichen Eckdaten dieser Implementierung sollen hier – soweit sie inhaltlich einen Beitrag leisten – kurz dargestellt werden. Es sei jedoch bemerkt, dass es sich hierbei um eine *prototypische* Implementierung handelt, die an dieser Stelle lediglich als *Proof-of-Concept* verstanden werden soll und sich an der von Albert Einstein ausgegebenen Maxime *"Keep it as simple as possible (but not simpler)!"* orientiert.

Die prototypische Implementierung zeigt, dass bereits eine Kombination aus

- Perl,
- Shell-Skripten in Verbindung mit *Makefiles* und

- das Open-Source-Kommandozeilentool *XML-Starlet*[11]

zur Zielerreichung ausreichen würde. Eine abstraktere, technischere und für den weiteren Verlauf dieser Arbeit wesentliche Implementierung des WebTool-Gedankens wird im nächsten Abschnitt (3.5, Seite 60ff.) vorgestellt.

Für die hier interessierende kommandozeilenorientierte Implementierung wurde Perl als Programmiersprache gewählt, weil diese Sprache sehr mächtig im Umgang mit Strings – und somit im Umgang mit den Inhaltsfragmenten – ist. Die im technischen Konzept genannten Validierungs- und Transformationsmechanismen für XML-Dateien stellt das Tool "XMLStarlet" zur Verfügung. Operationen auf dem Dateisystem (z.B. das Kopieren der generierten Dateien auf den Webserver) sowie die Koordination des Perl-Tools mit XML-Starlet lässt sich sehr einfach mit Hilfe elementarer Shell-Skripte sowie einem auf *Makefiles* basierenden Build-Management steuern und automatisieren. Alternativ zur Verwendung des Kommandozeilentools XMLStarlet steht der gebotene Funktionsumfang auch in Form von Perl-Modulen zur Verfügung, sodass auch eine Validierung und Transformation unmittelbar aus dem Programmcode heraus möglich ist. Auf die praktische Verwendung des WebTools hat diese Unterscheidung aber keine Auswirkungen.

Das WebTool setzt sich aus drei Säulen, die als Verzeichnisse repräsentiert sind, zusammen:

prg: enthält alle Skripte und Programme zur Generierung des Webauftritts

templates: enthält XML-Stylesheets und Dokument-Definitionen, die zur Validierung des Inhalts benötigt werden

src: enthält die Verzeichnisstruktur mit sämtlichen für den Inhalt relevanten Quelldateien sowie Konfigurationsoptionen; es entspricht dem in der Beschreibung erwähnten Wurzelverzeichnis.

3.4.4.1 Das Verzeichnis `templates`

In diesem Katalog befinden sich zentral alle Vorlagen, aus denen die Webseiten erzeugt werden. Dies betrifft einerseits die Layout-Vorlagen-Datei (`template.html`) nebst jeglicher begleitender Dateien (Bilder, CSS, etc.) sowie andererseits die Dokumenttyp-Definitionen der XML-Dateien und den zugehörigen XML-Stylesheets, mit deren Hilfe die datenzentrierten Seiten automatisch generiert werden.

3.4.4.2 Das Verzeichnis `src`

Für die Funktionsweise des WebTools von besonderem Interesse ist der Inhalt des Verzeichnisses `src`. Aus seiner Substruktur wird die Struktur des gesamten Webauftritts gewonnen. Insbesondere dient die Substruktur auch zur automatischen Erzeugung der Navigationselemente.

Dieser Katalog enthält die Quellen für die gesamte Website. Die Verzeichnisstruktur innerhalb dieses Katalogs wird auf das Zielverzeichnis auf dem Webserver exakt abgebildet. Die Unterkataloge, die unmittelbar in `src` liegen, bilden dabei die oberste Navigationsebene. In

[11] `http://xmlstar.sourceforge.com`

Tabelle 3.1: Liste der Konfigurationsoptionen

Eigenschaftsname	Bedeutung	Mehrfachnennung
title	Titel der momentanen Navigationsebene	Nein
navEntry	Eintrag in die Navigationsleiste der aktuellen Hierarchieebene	Ja
copyAsIs	Name eines Unterkatalogs oder einer einzelnen Datei, welche(r) nur kopiert wird; eine Einbeziehung in die Generierung findet nicht statt.	Ja
author	Bezeichnung eines speziellen Autoren einer Seite im aktuellen Katalog; ist kein Autor genannt, erfolgt der Default-Eintrag "Web-Administration".	Ja

den Unterkatalogen erfolgt dann jeweils ein rekursiver Abstieg zum Aufbau der gesamten Navigationsstruktur.

In jedes Verzeichnis unterhalb von `src` können beliebige Dateien abgelegt werden. Wie mit diesen bei der Generierung verfahren wird, steuern die jeweiligen Einträge in der zugehörigen Konfigurationsdatei. Deshalb muss jedes Unterverzeichnis innerhalb der `src`-Hierarchie die im folgenden Abschnitt beschriebenen Konfigurationsoptionen in Form einer Datei namens `config.cf` enthalten. Wenn diese Datei nicht gefunden werden kann, wird das aktuelle Verzeichnis bei der Generierung übersprungen.

3.4.4.3 Die Konfiguration des WebTools

Das WebTool benötigt Konfigurationsinformationen über jede Hierarchieebene. Diese dienen einerseits dazu, aus ihnen die Navigationsstrukturen aufzubauen und somit eine manuelle Pflege dieser Strukturen zu vermeiden. Andererseits werden sie verwendet, um Dateien und Verzeichnisse, die generiert werden sollen, von jenen zu unterscheiden, die nur für ein Kopieren vorgesehen sind. Die prototypische Implementierung verfährt dazu wie folgt:

Auf jeder Hierarchieebene (d.h. in jedem beliebigen Verzeichnis der Struktur) befindet sich eine Konfigurationsdatei namens `config.cf`. Es handelt sich dabei um jeweils eine reine Text-Datei, welche die Möglichkeit bietet, Parameter zu konfigurieren, die sich jeweils nur auf die aktuelle Hierarchie erstrecken. Wenn diese Datei in einem Verzeichnis nicht gefunden werden kann, wird das Verzeichnis bei der Generierung übersprungen.

Der Aufbau der Konfigurationsdateien gestaltet sich in einer zeilenweisen Auflistung von Schlüssel-Werte-Paaren, wobei der Schlüssel für eine Eigenschaft und der Wert für die Konfiguration dieser Eigenschaft steht. Je nach Schlüssel kann als Wert auch eine Liste von Werten aufgeführt werden. Eine Übersicht über die bekannten Eigenschaften gibt Tabelle 3.1.

Die dritte Spalte der Tabelle 3.1 gibt an, ob die jeweiligen Eigenschaft nur einmalig pro Konfigurationsdatei vorkommen darf oder ob eine Mehrfachnennung erlaubt ist. Die Notwendigkeit einer Mehrfachnennung der Eigenschaften `author` und `copyAsIs` ist deshalb gegeben, weil sich diese Eigenschaften jeweils auf eine von prinzipiell beliebig vielen regulären Dateien bzw. Verzeichnissen in der aktuellen Ebene beziehen. Grundsätzlich ebenfalls beliebig viele Einträge sind in den Navigationsstrukturen möglich, weshalb auch die Eigenschaft `navEntry` eine Mehrfachnennung erlaubt. Die Angabe eines Titels (Eigenschaft `title`) ist pro Ebene jedoch nur einmalig möglich.

Listing 3.1: Beispiel für eine Konfigurationsdatei

```
title;Lehre
navEntry;ss2009#Vorlesungen im Sommersemester 2009#Lectures in summer term 2009
navEntry;/sai/vorlesungen#Gesamtverzeichnis der Vorlesungen#All Lectures
copyAsIs;pics
author;index.html#Norbert Heidenbluth
```

Beispiel einer Konfigurationsdatei: Listing 3.1 zeigt ein Beispiel für eine Konfigurationsdatei. Die einzelnen Inhalte haben dabei die folgende Bedeutung:

- Der Titel der augenblicklichen Navigationsstruktur lautet "Lehre".
- Es werden zwei Einträge (Links) in der aktuellen Ebene der Navigationsstruktur erzeugt: einer für die Vorlesungen im Sommersemester 2009 und ein anderer zum Verzeichnis aller Vorlesungen. Der Aufbau des Schlüssels der Eigenschaft `navEntry` besteht dabei aus dem Ziel des Links (relativ), dem Eintrag in das Menü in deutscher Sprache sowie dem Eintrag in das Menü in englischer Sprache; diese Angaben werden jeweils durch das Symbol "#" voneinander getrennt (siehe zur Mehrsprachigkeit auch Abschnitt 3.4.3.7).
- Der Unterkatalog `pics` wird so, wie er vorliegt, in das Webverzeichnis kopiert. Sein Inhalt wird *nicht* weiter zum Zwecke einer Generierung durchlaufen.
- Als Autor der Seite `index.html` wird "Norbert Heidenbluth" genannt. Auch hier gilt wieder ein Aufbauschema: dem Namen der generierten Datei folgt – getrennt durch das Zeichen "#" – der Name ihres Autors (bzw. eine Aufzählung der Autoren).

Der letztgenannte Punkt (Nennung des Autors bzw. der Autoren) könnte dahingehend verbessert werden, dass er aus den Konfigurationsdateien ausgelagert und stattdessen in den jeweiligen Inhaltsfragmenten verankert wird. Dann stünden die entsprechenden Informationen unmittelbar dort, wo sie vermutlich auch erwartet werden: bei den betreffenden Inhalten selbst.

3.4.4.4 Beispiel für die Anwendung des WebTools

Es wurde bereits erwähnt, dass die prototypische Implementierung des WebTools seit Sommer 2007 zur Unterhaltung eines realen Webauftritts im Einsatz ist. Abbildung 3.3 zeigt eine Seite, die auf die beschriebene Art generiert wurde.

Der mit "1." markierte Bereich spiegelt die thematische Sortierung des Webauftritts wider: jeder Reiter entspricht einem Unterverzeichnis innerhalb des Wurzelkatalogs (`src`). Diese Hauptnavigationsleiste (Navigationsstruktur erster Ordnung; *Top-Level-Navigation*) wird auf jeder generierten Seite auf diese Weise dargestellt. Der jeweils aktuell gewählte Bereich ist dabei zum Zwecke der Erleichterung der Orientierung farblich hervorgehoben (invertiert). Der mit "2." markierte Bereich enthält die Navigationsstruktur zweiter Ordnung. Diese ist abhängig vom momentan aktiven Bereich und dient zur Navigation innerhalb einer Hierarchieebene. Alle Navigationselemente von Hierarchietiefen ≥ 2 werden in diesem Bereich abgebildet. Beide Navigationsbereiche werden vom Tool dynamisch erzeugt und erfordern keine manuelle Bearbeitung. Sie ergeben sich aus der Verzeichnisstruktur unterhalb von `src` sowie aus den Inhalten der jeweiligen Konfigurationsdateien.

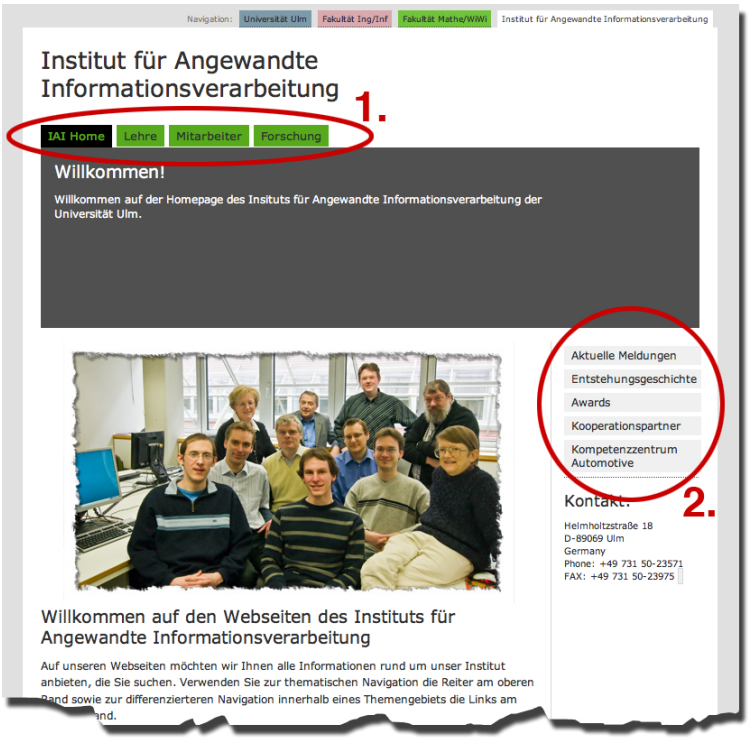


Abbildung 3.3: Screenshot der Begrüßungsseite des IAI (Stand 05/2009)

Die zwei Bereiche, die als Container für den eigentlichen Inhalt bereit stehen, zeigt Abbildung 3.4. Der grau unterlegte Bereich ("3.") nimmt eine Kurzzusammenfassung des Seiteninhalts auf, und Bereich Nr. 4 enthält dann den eigentlichen Seiteninhalt. Der Bereich 4 ist zur Darstellung der HTML-Inhaltsfragmente vorgesehen. Wenn es sich um datenzentrierten Inhalt handelt, werden die HTML-Inhaltsfragmente automatisch erzeugt. Dieses trifft auch zu, wenn eine eigene Auszeichnungssyntax (nebst Transformator) zum Einsatz kommt. Lediglich im Falle freier Inhalte, die unmittelbar in HTML erfasst werden, beeinflusst der Benutzer somit indirekt die Qualität des späteren HTML-Codes. Positiv ausgedrückt bedeutet dies, dass im Idealfall die gesamte Seite generiert werden kann, aber auch im Falle manueller HTML-Inhalte der größte Teil der endgültigen Seite das Produkt einer automatischen Generierung ist.

Um anhand dieses konkreten Einsatzszenarios des WebTools nochmals die Verwendung der Konfigurationsdatei zu verdeutlichen, zeigt Programmtext 3.2 die vollständige Konfigurationsdatei, welche zur in Abbildung 3.4 gezeigten generierten Seite geführt hat.

Listing 3.2: Beispiel für eine Konfigurationsdatei

```
1  title;IAI Home
2  author;index.html#Norbert Heidenbluth
3  author;historie.html#Franz Schweiggert
4  author;partner.html#Franz Schweiggert
5  copyAsIs;pics
6  copyAsIs;downloads
7  navEntry;aktuelles.html#Aktuelle Meldungen
8  navEntry;historie.html#Entstehungsgeschichte
9  navEntry;awards.html#Awards
10 navEntry;partner.html#Kooperationspartner
11 navEntry;http://automotive.uni-ulm.de#Kompetenzzentrum Automotive
```

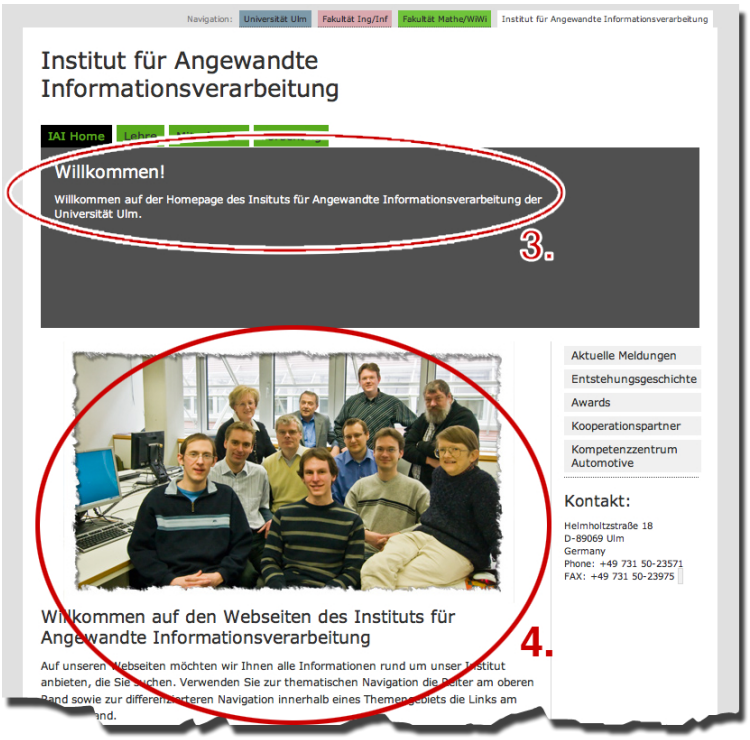


Abbildung 3.4: Screenshot der Begrüßungsseite des IAI (Stand 05/2009)

Erläuterung zum Codebeispiel 3.2

Zeile 1: Dieser Eintrag gibt den Titel der gesamten Hierarchie-Ebene an. Da das Beispiel die Begrüßungsseite des IAI-Webauftritts zeigt, entspricht die aktuelle Ebene der ersten. Mithin ist der Titel zugleich auch die gewählte Rubrik im Top-Level-Navigationsmenü.

Zeilen 2-4: Hier können (optional) Informationen über die Autoren einzelner Seiten hinterlegt werden. Diese Informationen werden – sofern vorhanden – in die Fußzeile der Seiten hineingeneriert. Liegen Informationen über die Autoren nicht vor, so erfolgt der automatische Eintrag "Web-Administration".

Zeilen 5 und 6: Die Unterordner `pics` und `downloads` sollen lediglich auf das Zielverzeichnis *kopiert* werden, ansonsten aber beim rekursiven Durchlaufen zum Zwecke der Generierung unberücksichtigt bleiben.

Zeilen 7-11: Diese Einträge geben die Inhalte des zweiten Navigationsbereichs (hier: am rechten Rand) an. Mit Ausnahme des Eintrags in Zeile 11 verweisen die generierten Links auf Seiten innerhalb derselben Navigationsebene, der Link aus Zeile 11 hingegen führt zu einer externen Seite.

3.4.5 Validierung und Transformation von XML-Daten

Die schon mehrfach erwähnte automatische Validierung sowie die Transformation von XML-Dokumenten sollen wegen ihrer Bedeutung für den weiteren Verlauf dieser Arbeit hier noch eine ausführlichere Erklärung erfahren. Die grundlegende Idee ist in Abbildung 3.5 skizziert.

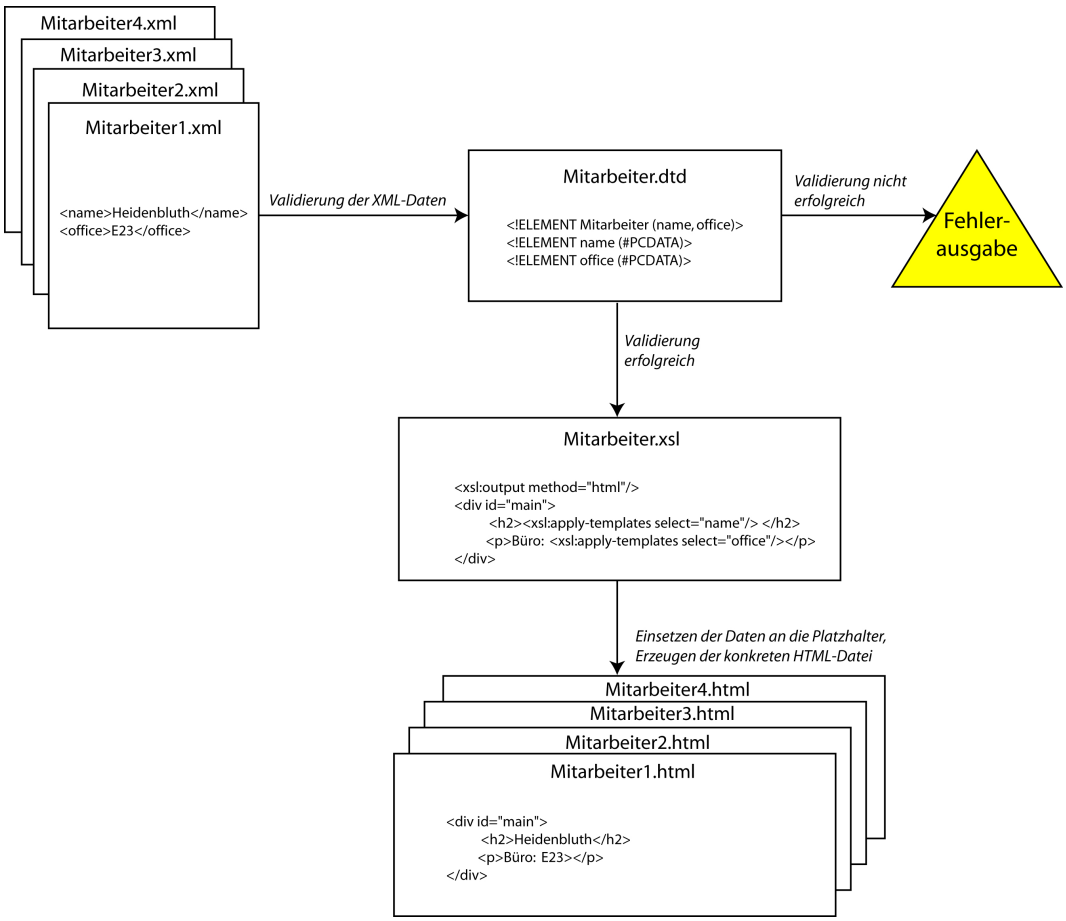


Abbildung 3.5: Ablauf der XML-Transformation im WebTool (hier: IAI-Mitarbeiterseiten)

XML-Dokumente repräsentieren in der Regel Daten in einer fest vorgegebenen Struktur. Eine solche Struktur ist frei bestimmbar und kann in einer separaten Datei beschrieben werden. Ein XML-Dokument führt dann die Information mit sich, welcher Struktur(datei) sein Aufbau folgt, und mithin lässt sich einfach überprüfen, ob ein konkretes XML-Dokument einen gültigen Aufbau hat.

Beispiel: Manche Seiten innerhalb eines Webauftritts weisen eine stets gleiche inhaltliche Struktur auf: innerhalb des IAI-Webauftritts sind dies beispielsweise die Mitarbeiterseiten sowie die Seiten zu Vorlesungen und Seminaren, aber auch die Ankündigungen von zu vergebenden Abschlussarbeiten. Diese Seiten beschreiben stets genau einen Mitarbeiter bzw. genau eine Vorlesung, ein Seminar oder eine Abschlussarbeit. Die inhaltlichen Informationen sind aber innerhalb eines Genres (z.B. “Mitarbeiter”, “Vorlesung” oder “Abschlussarbeit”) stets identisch. Somit kann einmalig zentral der Aufbau der XML-Beschreibung eines Mitarbeiters, einer Vorlesung oder einer Abschlussarbeit vorgenommen werden, und die entsprechenden Inhalte folgen dann stets dieser Beschreibung.

Grundsätzlich kommen für die Beschreibungen nebst Validierungen zwei Ansätze in Frage:

- Dokumenttypdefinitionen (englisch: *Document Type Definition*, DTD) und
- XML-Schemata

Beide Ansätze verfolgen ein identisches Ziel: sie beschreiben die Struktur einer Klasse von XML-Dokumenten. Während DTD-Dateien eine eigene Syntax für die Beschreibung der Strukturregeln verwenden, sind XML-Schemata ihrerseits selbst auch XML-Dateien. Sie bedienen sich somit eines anderen Sprachmittels als DTD-Dateien und ermöglichen dadurch eine deutlich höhere Flexibilität in der Strukturbeschreibung.

Mit Hilfe entsprechender Tools kann jede beliebige XML-Datei gegen eine Typdefinition (bzw. ein Schema) validiert werden. Mithin lässt sich also überprüfen, ob eine gegebene XML-Datei einen gültigen Aufbau aufweist. Für den Einsatz innerhalb des WebTools ist eine Unterscheidung zwischen Dokumenttypdefinitionen und XML-Schemata aber nicht von Bedeutung: das WebTool erlaubt die Verwendung beider Ansätze.

Valide XML-Dokumente können in einem zweiten Schritt dann in beliebige andere Formate transformiert werden. Auch dazu liegt eine entsprechende Beschreibungsdatei vor, welche konfiguriert, wie eine solche Transformation vorgenommen werden muss. Derartige Beschreibungsdateien liegen unter der Bezeichnung *Extensible Stylesheet Language Transformations* (XSLT) in einer eigenen Sprache vor, welche ihrerseits wieder auf XML basiert und durch entsprechende Prozessoren die Transformationen von XML-Dokumenten in Zielformate vornehmen kann.

Der Vorteil eines solchen Vorgehens ist der Erhalt der Gültigkeit mit jedem Verarbeitungsschritt. Abbildung 3.5 skizziert den Verarbeitungsprozess des WebTools von datenzentrierten Inhalten vom Zeitpunkt der Entstehung der XML-Dateien bis hin zur Generierung der endgültigen Seite. Unter der Voraussetzung, dass die Vorlagendatei (`template`) und die Transformationsdatei gültig sind (dies muss einmalig überprüft werden), ergibt sich somit die Gültigkeit aller korrespondierenden Inhaltsseiten. Gleiches gilt für die Barrierefreiheit: berücksichtigt die Transformationsdatei eine barrierefreie Ausgabe, so entstehen daraus auch unmittelbar barrierefreie Zieldateien.

Auch die Unterstützung der Mehrsprachigkeit lässt sich durch den Einsatz von Validierung und Transformation bequem realisieren. Hierzu sind zwei technische Ansätze denkbar:

unilinguale XML-Dokumente: Jedes XML-Dokument beinhaltet Daten für jeweils nur genau eine Sprache. Entsprechend existiert für jede Sprache eine zugehörige Dokumenttyp-Datei (respektive ein Schema) sowie eine Transformations-Datei. Weil in XML-Dokumenten Informationen über "ihren" Dokumenttyp und die erforderliche Transformationsdatei enthalten sein können, kann jedes XML-Dokument automatisiert den der jeweiligen Sprache entsprechenden Validierungs- und Transformationsprozess durchlaufen. Der Nachteil ist hier jedoch, dass für jede Sprache eine eigene XML-Datei vorhanden sein muss, was den Pflegeaufwand mit jeder unterstützten Sprache erhöht und eine hohe Redundanz in der Datenhaltung auslöst. Der Vorteil dieses Ansatzes liegt hingegen darin, dass sich auf nur eine Sprache optimierte XML-Dokumente unmittelbar transformieren lassen.

multilinguale XML-Dokumente: Ein aus der Sicht der Datenhaltung her wesentlich besserer Ansatz ist die Zusammenfassung aller Daten (auch in verschiedenen Sprachen) in nur einem XML-Dokument, insbesondere deshalb, weil es dadurch keine Redundanzen mehr gibt. Hinsichtlich der Validierung macht dieses Vorgehen keinen Unterschied zum unilingualen Ansatz. Bei der Transformation hingegen macht es eine Fallunterscheidung nötig, die im regulären Transformationsprozess zunächst einmal nicht vorgesehen ist. Für jede unterstützte Sprache muss eine eigene Transformation durchgeführt werden, ein XML-Dokument kann aber nur *eine* Information über die zugehörige Transformationsdatei aufnehmen. Mithin ergeben sich auch hier zwei verschiedene Verfahrensweisen:

- Die Transformationsdatei enthält Transformationsregeln für *alle* unterstützten Sprachen zugleich. Weil XSLT ähnlich wie eine Programmiersprache verwendet werden kann, erlaubt es Fallunterscheidungen. Somit kann die Transformation eines multilingualen XML-Dokuments mit derselben XSLT-Datei für jede Sprache vorgenommen werden, sofern vor Beginn der Transformation das XML-Dokument um die Angabe der gerade aktuellen Sprache ergänzt wird. Mit Hilfe dieser zusätzlichen Information kann dann die Fallunterscheidung in der XSLT-Datei vorgenommen und eine sprachspezifische Transformation erreicht werden.
- Für jede unterstützte Sprache existiert genau eine XSLT-Datei, welche sich ausschließlich auf diese Sprache bezieht. Das XML-Dokument wird vor der Anwendung der Transformation dahingehend manipuliert, dass die in ihm enthaltene Information über die zugehörige Transformationsdatei entsprechend der aktuellen Sprache verändert wird.

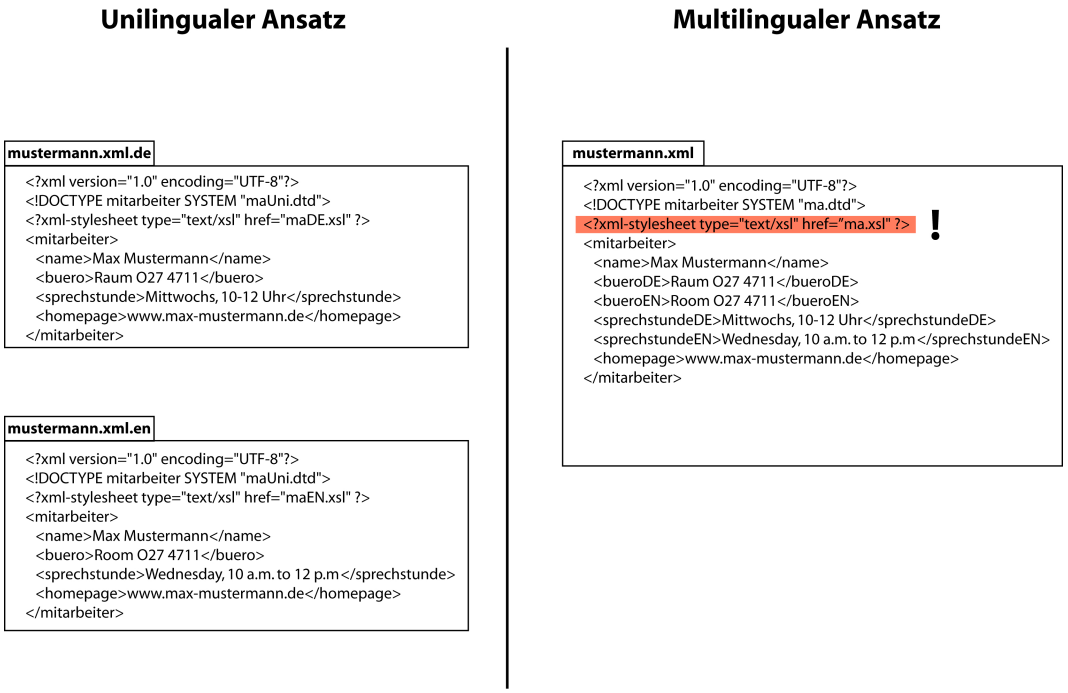


Abbildung 3.6: Unterschiede zwischen uni- und multilingualen XML-Dokumenten

Beispiel: Abbildung 3.6 visualisiert die beiden beschriebenen Ansätze und ihre Nachteile: im unilingualen Ansatz genügt die Bereithaltung einer Dokumenttyp-Datei (respektive eines Schemas), für jede unterstützte Sprache muss jedoch ein *vollständiges* XML-Dokument erstellt werden. Sprachunabhängige Elemente (im Beispiel: Name und Homepage) werden auf diese Weise aber redundant vorgehalten. Die Transformation der XML-Dokumente ist ebenfalls sprachabhängig möglich; dazu genügt die Angabe der jeweiligen (sprachspezifischen) XSLT-Datei (vgl. die dritte Zeile der Beispieldateien). Die multilinguale Vorgehensweise hält alle Informationen in nur einer XML-Datei vor. Dies vermeidet unnötige Redundanzen, jedoch lässt sich nun nicht mehr unmittelbar eine Transformationsdatei verwenden. Vielmehr müssen verschiedene derartige Dateien zur Anwendung kommen, um aus den Daten jeweils eine spezielle Sprachfassung der Zieldatei zu generieren. Der in der Abbildung hervorgehobene Teil muss deshalb dynamisch durch die einzelnen sprachspezifischen Dateien ersetzt werden.

Denkbar wäre hier auch eine Mischform aus sowohl unilingualem als auch multilingualem Ansatz: in einer Datei stünde dann alles, was sprachunabhängig ist (das "Gemeinsame"), und in einer anderen würden dann *nur* die sprachabhängigen Informationen (separat nach Sprache) vorgehalten.

3.4.6 Diskussion des WebTool-Einsatzes

Die Darstellung der grundlegenden Idee des WebTools sowie dessen prototypischer Implementierung soll mit einer kurzen Diskussion über Stärken und Schwächen dieses Ansatzes schliessen.

3.4.6.1 Stärken

Zu den besonderen Stärken dieses Ansatzes zählt insbesondere, dass er auf verschiedene, etablierte und hinreichend stabil verfügbare Werkzeuge und Technologien zurückgreift. Er führt einige der im Web- und XML-Umfeld gängigen Technologien zusammen und kombiniert sie zu einem Gesamtkonstrukt. Gerade die Tatsache, dass jede Komponente für sich genommen nicht neu ist, sondern als etabliert bezeichnet werden kann, garantiert dem Ansatz eine Stabilität im technischen Fundament. Neuartig ist hingegen die Art und Weise, *wie* die Kombination der Techniken erfolgt. Während die bisherige Beschreibung diesbezüglich anhand existierender Tools erfolgte, wird der folgende Abschnitt eine abstraktere Sicht auf das WebTool zum Inhalt haben und auf die Bereitstellung des WebTools in Form eines *Frameworks* führen, das nicht mehr kommandozeilenbasiert, sondern modular z.B. in einem Applikationsserver betrieben werden kann.

Für die Anwender liegen die Stärken des WebTools darin, dass viele Erwartungen, die an ein Content-Management-System gestellt werden, auch durch das WebTool erfüllt werden können. Dabei tritt es aber nicht als in sich geschlossene Softwarelösung auf, sondern als kommandozeilenbasierter Generator. Anforderungen an den Mehrbenutzerbetrieb und die damit verbundene Vergabe von Berechtigungen können in gewohnter Weise mit Hilfe von durch das Betriebssystem bereitgestellten Mechanismen erfüllt werden. Jeder Benutzer (hier: Redakteur oder Autor) kann zum Bearbeiten und Erstellen von Inhalten "seinen" gewohnten Editor verwenden und ist in der Gestaltung seiner Arbeitsweise in keiner Weise den Grenzen eines Content-Management-Systems unterworfen. Unterschiedlichen Kenntnissen im Umgang mit HTML kann durch verschiedene Module, die bei der Generierung der Ausgabeseiten verwendet werden, Rechnung getragen werden. Insbesondere kann auch auf die Verwendung von HTML vollständig verzichtet werden und der Inhalt stattdessen auch in Form von Text-Dateien vorliegen, die einer anderen Auszeichnungssprache als HTML (z.B. die in [Wik] vorgestellte Wiki-Syntax) oder gar einer selbstdefinierten Syntax folgen. Sofern ein Transformationsmodul bereitgestellt wird, kann jede beliebige Beschreibungssprache verwendet werden. Mithin lässt sich der Inhalt eines mit dem WebTool erzeugten Webauftritts auch durch Personen pflegen und erweitern, welche keine Kenntnis von HTML besitzen. Ebenso werden vom Benutzer keine Layout-Kenntnisse erwartet, weil die Trennung von Layout und Inhalt eine ganz zentrale Anforderung an das WebTool ist. Im Gegenteil: es ist im Sinne der Einhaltung von Standardkonformität und Optimierung der Barrierefreiheit ausdrücklich *nicht* erwünscht, dass die Redakteure Inhalt mit Layout mischen.

Die Autoren des Inhalts sind aber nicht nur frei von Verantwortung für das Layout, sondern darüber hinausgehend auch frei von Überlegungen zur Barrierefreiheit. Sofern die vom WebTool verwendete Vorlagen-Datei barrierefrei ist (dies muss einmalig sichergestellt sein),

überträgt sich diese Eigenschaft auch auf alle generierten Seiten, insbesondere dann, wenn eine andere Auszeichnungssprache als HTML zum Einsatz kommt. Dann nämlich unterliegen die Autoren des Inhalts nicht mehr der Gefahr, durch unbedachten Umgang mit HTML-Elementen Barrieren in die Quellen einzuschleusen.

Auch die beschriebene Anwendung von XML-Transformationen trägt zur Barrierefreiheit und darüberhinaus zur Qualitätskontrolle bei. Weil die Validierung der XML-Daten gegen die DTD-Datei vorgenommen wird (vgl. Abbildung 3.5) und auf nicht valide XML-Vorlagen kontrolliert reagiert werden kann, liegen nur inhaltlich vollständige und korrekte Daten zur Weiterverarbeitung vor. Die Verantwortung für die Barrierefreiheit der aus den XML-Dateien erzeugten HTML-Version obliegt dann noch ausschließlich dem Autor des XML-Stylesheets (XLST-Datei), nicht jedoch den Autoren des XML-Inhalts (welcher ja seinerseits wieder generiert worden sein kann, z.B. aus Datenbanken).

Darüberhinaus sei auch noch die Plattformunabhängigkeit des WebTools erwähnt. Denn da es im Wesentlichen auf der Programmiersprache Perl basiert, lässt es sich über verschiedene Betriebssysteme hinweg ohne nennenswerte Anpassungen verwenden. Das WebTool wurde auf verschiedenen Unix-artigen Betriebssystemen entwickelt, sodass bei Bedarf die plattformabhängigen Inhalte in den Makefile-Dateien an Windows-eigene Vorgehensweisen adaptiert werden müssen.

3.4.6.2 Erfüllung der ATAG-Richtlinien

Die Anforderung 3a (Seite 40) erwartet von dem WebTool die Einhaltung möglichst vieler der ATAG-Richtlinien (siehe Seite 33). Deshalb soll an dieser Stelle eine Evaluierung vorgenommen werden, inwieweit das WebTool die Einhaltung dieser Richtlinien unterstützt.

1. Unterstütze barrierefreie Arbeitsabläufe der Autoren!
   Erfüllt! Weil das WebTool kein in sich geschlossenes Software-System ist, innerhalb dessen die Benutzer arbeiten, steht jedem Benutzer frei, die Werkzeuge (Texteditoren) seiner Wahl zu verwenden.

2. Erzeuge standardisierte Ausgaben!
   Erfüllt! Sofern die Vorlagen valide (standardisiert) sind, folgt daraus auch eine valide Ausgabe (vgl. dazu auch Abbildung 3.6).

3. Unterstütze die Erstellung barrierefreien Inhalts!
   Erfüllt! Das WebTool ist insbesondere auf die Erzeugung barrierefreier Webauftritte ausgerichtet.

4. Biete Möglichkeiten zum Überprüfen und Korrigieren von nicht-barrierefreiem Inhalt an!
   Erfüllt! Die Validierung der bei der Generierung zu berücksichtigenden Inhalte ist die zentrale Idee des WebTool-Ansatzes.

5. Integriere Lösungen zur Barrierefreiheit in das gesamte Erscheinungsbild (der Software)!
   Nicht erfüllt! Weil das WebTool die größtmögliche Freiheit in der Verwendung der Tools gewährt, kann dieser Punkt nicht realisiert werden.

6. Mache in den Hilfe-Texten und in der Dokumentation auf Barrierefreiheit aufmerksam!
   Nicht erfüllt! (vgl. Erklärung zu Punkt 5; eine Einflussnahme auf die Hilfe-Texte und Dokumentationen der frei wählbaren Tools ist nicht möglich.) Hingegen kann das WebTool beim Fehlschlagen von Validierungen ein Hinweis geben, weshalb die Validierung

nicht erfolgreich verlief, d.h. was genau die beklagte Standard-Verletzung ausmacht. Dies kann als ein Beitrag zur Erfüllung dieses Punktes der ATAG-Richtlinien angesehen werden.

7. Stelle sicher, dass das Autorensystem auch für Menschen mit Behinderungen zugänglich ist.
   Erfüllt! (vgl. Erklärung zu Punkt 1)

Mithin erfüllt das WebTool unmittelbar fünf der sieben ATAG-Richtlinien. Die nicht erfüllten Empfehlungen liegen hingegen außerhalb des Einflussbereichs des WebTools.

3.4.6.3 Schwächen

Die Simplizität des WebTool-Ansatzes birgt auch Nachteile, durch welche insbesondere die Abgrenzung zu einem vollwertigen Content-Management-System demonstriert wird:

- Das WebTool sieht keine unterschiedlichen Benutzerrechte oder Benutzerrollen vor; die Berechtigungen, Dateien zu ändern ergeben sich unmittelbar aus den Rechten innerhalb des Dateisystems.

- Eine Bearbeitungshistorie der Seiten ist nicht nativ vorgesehen. Selbstverständlich kann dies aber leicht durch den Einsatz von externen Versionsverwaltungstools wie beispielsweise CVS[12] oder SVN[13] dennoch realisiert werden.

- Der erzeugte Webauftritt beschränkt sich auf statische Seiten; ein Interaktionspotenzial im Sinne von dynamisch erzeugtem Inhalt ist nicht möglich.

- Der kommandozeilenorientierte Charakter und die Wahlfreiheit hinsichtlich der zu verwendenden Editoren haben zur Folge, dass sich das WebTool nicht mit einer einheitlichen grafischen Benutzerschnittstelle (GUI) präsentieren kann. Zwar kann der unmittelbare Umgang mit der Kommandozeile durch die Einführung einer beispielsweise auf Perl/TK basierenden GUI vermieden werden, jedoch gibt es nicht "das WebTool" als eigenständige Software-Komponente, die unter einer einheitlichen GUI alle bereitgestellten Funktionalitäten aggregiert. Dies ist allerdings *ausdrücklich nicht* vorgesehen, weil jeder Benutzer die Freiheit in der Wahl seiner "Werkzeuge" (z.B. Editoren) behalten soll.

3.4.6.4 Ein Fazit zur prototypischen Implementierung des WebTools

Zusammenfassend können für die prototypische Implementierung des WebTools (IAI-WebTool) die folgenden Vorteile festgehalten werden:

- Es bietet eine gute Skalierbarkeit: weil die Erzeugung der Seiten nur auf dem rekursiven Auslesen bzw. Verarbeiten einer Verzeichnisstruktur beruht, sind prinzipiell beliebig tiefe und verschachtelte Seitenstrukturen möglich.

[12] CVS = Concurrent Version System, siehe `http://www.nongnu.org/cvs/`
[13] SVN = Subversion, siehe `http://subversion.tigris.org/`

- Bei reinen datengesteuerten Seiten (im Beispiel: Vorlesungsverzeichnis, Mitarbeiterübersicht) führt die Verwendung von XML-Transformationen zugleich zu einer einfachen, aber effektiven Qualitätskontrolle: durch Validierung der XML-Dateien gegen ihre zugehörige DTD-Datei (alternativ: gegen ein Schema) wird sichergestellt, dass der zu erzeugende Seiteninhalt (d.h. die Daten) zur Vorlage passt.
- Die einzelnen Autoren müssen sich nicht um Barrierefreiheit und Layout kümmern. Dies wird nur einmalig zentral beim Erstellen der Vorlagen-Datei gemacht.
- Durch Austauschen der CSS-Dateien kann der generierte Inhalt in verschiedenen Versionen dargeboten werden (z.B. auch besonders kontrastreich, besonders groß, etc.).

In der bislang beschriebenen Form eignet sich das WebTool für den Einsatz für informierende Webauftritte, wobei der Großteil des Inhalts idealerweise in Form von standardisierten XML-Dateien vorgehalten und wie zuvor beschrieben nach HTML transformiert werden kann. Weil die Seiten so lange in unveränderter Form im Dateisystem (des Webservers) liegen, bis eine erneute Generierung veranlasst wird, ist das WebTool eher für nicht-flüchtige Inhalte (d.h. Inhalte, die sich nicht kurzfristig ändern) zu empfehlen.

Die mit dem WebTool verfolgten Ideen lassen sich gemäß den Ausführungen in diesem Kapitel bereits durch eine geeignete Kombination vorhandener Perl-Bibliotheken und Shell-Kommandos dergestalt umsetzen, dass die wichtigsten der ATAG-Richtlinien erfüllt werden und die Benutzer dabei die größtmögliche Freiheit in der Wahl ihrer Arbeitsumgebung erhalten. Die Ideen hinter dem WebTool können aber nicht nur als kommandozeilenbasierte Stand-Alone-Lösung realisiert werden. Sie lassen sich vielmehr abstrahieren, in einzelne funktionale Bestandteile (Module) gliedern und somit in ein *Framework* überführen. Der folgende Abschnitt stellt daher die Implementierung des WebTools als Framework vor. Dieses dient dann zugleich auch als Grundlage für die nachfolgenden Kapitel dieser Arbeit.

3.5 jLAF

3.5.1 Motivation

Das zuvor beschriebene WebTool stellt eine funktionsfähige und effektive Implementierung der zugrunde liegenden Konzepte dar. Als Stand-Alone-Lösung kann es – wie geschildert – bequem eingesetzt werden, jedoch setzt die Tatsache, dass das WebTool auf Perl und Shell-Skripten basiert, schnell Grenzen im Hinblick auf seine Integrierbarkeit in Web- oder Applikationsserver. Um das angestrebte Ziel (Generierung von HTML-Fallback-Lösungen von statischen RIAs) zu erreichen, ist eine solche Integration jedoch von wesentlicher Bedeutung.

Aus diesem Grunde werden die beschriebenen Konzepte des WebTools nun in eine objektorientierte (Java-)Welt übertragen. Hierbei handelt es sich jedoch nicht um ein bloßes Neu-Schreiben des Programmcodes in einer anderen Programmiersprache (Java), sondern vielmehr soll dabei eine Übertragung der Konzepte auf eine Lösung vorgestellt werden, die

- eine einfache Verwendung in Web- und Applikationsservern erlaubt,
- wahlweise eine statische oder dynamische ("on-the-fly") Seitengenerierung offeriert,
- die technische Grundlage für die Generierung von HTML-Fallback-Lösung statischer RIAs (siehe Kapitel 5) darstellt und

- durch Verwendung von objektorientierten Programmierprinzipien und Design-Pattern als Framework konzipiert ist und eine vielfältige Wiederverwendbarkeit erlaubt.

Im Rahmen dieser Arbeit entstand als entsprechende Lösung das "*Java Lightweight Authoring Framework*" (kurz: "jLAF"), das im folgenden genauer erläutert wird.

3.5.2 Überblick über das jLAF

Sämtliche im Zusammenhang mit dem WebTool bereits erläuterten Konzepte finden sich auch im jLAF wieder. Dies gilt insbesondere für das Bestreben, möglichst frühzeitig invalide Daten bzw. Dokumente zu erkennen sowie eine Handhabung auch für Benutzer mit nur geringen HTML- und Programmierkenntnissen zu ermöglichen. Der Einsatz des jLAF entspricht den Schritten 3 und 4 in Abbildung 3.2 auf Seite 43.

Die Realisierung des WebTool-Konzepts in einer objektorientierten Umgebung erweitert das bisherige Web-*Tool* (das "Werkzeug" zur Erstellung eines statischen Webauftritts) zu einem *Framework*. Dessen Verwendung ermöglicht sowohl die Generierung kompletter statischer Webauftritte als auch die Generierung einzelner Seiten dynamisch bei "Bedarf" ("*on the fly*"). Die Version des Frameworks, die im Rahmen dieser Arbeit entstanden ist, eignet sich für Webauftritte, deren Seitenstruktur derjenigen entspricht, die in Abbildung 3.1 auf Seite 42 vorgestellt wurde. Durch das Hinzufügen eigener Implementierungen von der im Framework enthaltenen abstrakten Klassen können aber auch andere (eigene) Wünsche an die Seitenstruktur realisiert werden. Die in diesem Zusammenhang relevanten Klassen des Frameworks werden im Abschnitt 3.5.4.4 genauer vorgestellt.

Genau wie das WebTool verwendet auch jLAF Konfigurationsdateien, mit denen sich Parameter wie beispielsweise Angaben zu den verwendeten Seitentemplates oder die Namen der Quell- und Zielverzeichnisse verwalten lassen. An die Stelle des eigenen Formats, wie es im WebTool verwendet wird, tritt nun lediglich die Steuerung über in Java übliche *Properties*-Dateien. Eine detailliertere Erklärung des Konfigurationskonzepts folgt in Unterabschnitt 3.5.3.

Das jLAF sieht in der in dieser Arbeit vorgestellten Version eine hierarchische Navigation über mehrere (theoretisch: beliebig viele) Ebenen vor. Dabei ist auf jeder generierten Seite stets eine *Top-Level*-Navigation vorgesehen sowie – darüber hinaus – eine auf die Einträge der aktuellen Ebene bezogene lokale Navigation. Diese enthält auch jeweils eine Verlinkung zur unmittelbar übergeordneten Ebene.

Die Intention des Frameworks ist, im Zusammenspiel mit den noch folgenden Erweiterungen des Kapitels 5 in einem Applikations-Server betrieben zu werden, denn für einen – wie im Falle des Webtools vorgesehenen – rein kommandozeilen-basierten Einsatz ist Java aufgrund des benötigen Aufwands zum Starten der virtuellen Maschine (JVM) nicht die beste Wahl. Hierzu steht mit der Perl-Version des WebTools bereits eine adäquate Lösung zur Verfügung.

Die wesentliche Unterscheidung zwischen dem Einsatz des jLAF für *statische* bzw. für *dynamische* Seitengenerierung zeigen die nachstehenden zwei Unterabschnitte.

3.5.2.1 Statische Generierung

Die Tatsache, dass das jLAF-Framework eine statische Generierung von Seiten bietet, scheint nicht recht zu der in der vorausgegangenen Motivation genannten Aussage, dass es zum Einsatz in einem Applikationsserver bestimmt ist, zu passen, denn ein Applikationsserver ist ja gerade

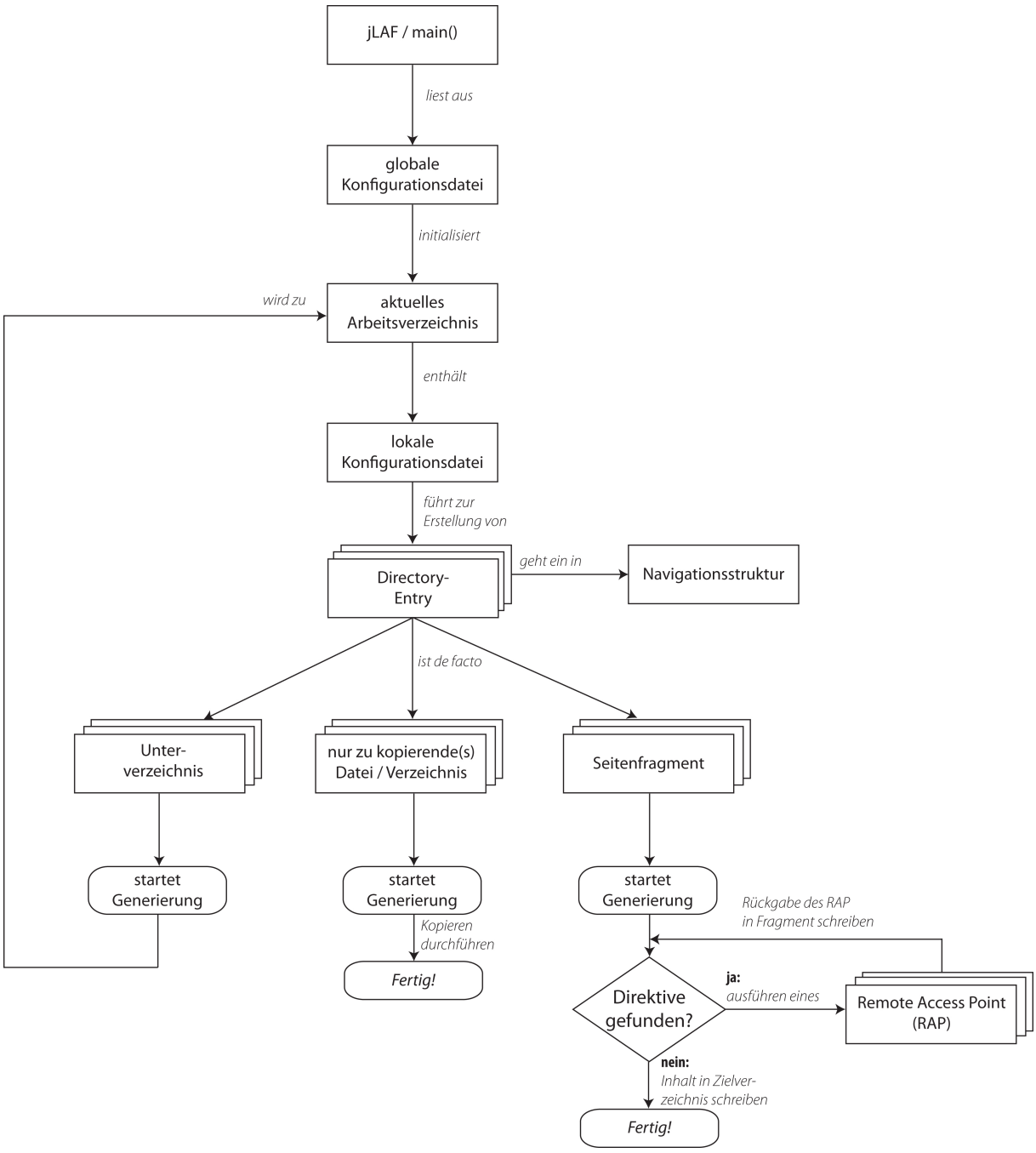


Abbildung 3.7: "*Big Picture*" — Die Funktionsweise des jLAF

dann im Einsatz, wenn es auf eine *dynamische* Seitengenerierung ankommt. Dieser scheinbare Widerspruch wird nach den Ausführungen in Kapitel 5 jedoch aufgelöst sein, denn dort wird der eigentliche Einsatzzweck des Frameworks betrachtet. Die hier aktuell beschriebene Verwendung für die Generierung statischer Seiten ist nur ein "Nebenprodukt". Anders formuliert: die Intention der Entwicklung von jLAF ist nicht nur die Generierung statischer Webseiten, sondern darüberhinaus vor allem dessen Verwendung in einem anderen (in Kapitel 5 erläuterten) Zusammenhang.

Der Ablauf beim Einsatz des jLAF zur Erzeugung statischer Webseiten ist in Abbildung 3.7 dargestellt. Zunächst wird die globale Konfigurationsdatei ausgewertet und somit der Generierungsdurchlauf initialisiert. Aus der globalen Konfiguration ergibt sich das Startverzeichnis, welches die lokalen Konfigurationsinformationen enthält. Diese geben an, welche Dateien bzw. Unterverzeichnisse generiert werden sollen und welche nur für eine reine Kopier-Operation vorgesehen sind. Als weitere Angabe enthalten die lokalen Konfigurationsinformationen die Beschriftungen, unter denen sich die jeweiligen Seiten bzw. Verzeichnisse in die Navigationsstrukturen einbringen. Somit erfolgt zunächst eine Erzeugung der Navigationsstruktur(en). Sodann wird für jeden Eintrag im aktuellen Verzeichnis (aufgrund der lokalen Konfigurationsinformationen) entweder dessen Generierung oder ein Kopieren veranlasst. Verzeichnisinhalte, über die in den Konfigurationsdateien keine Informationen enthalten sind, werden dabei (nebst Ausgabe einer entsprechenden Warnung) übersprungen. Für jedes enthaltene und zur Generierung vorgesehene (Unter-)Verzeichnis erfolgt ein rekursiver Aufruf des gerade beschriebenen Algorithmus. Somit ist der Ablauf im wesentlichen identisch mit demjenigen, der bereits im Zusammenhang mit dem WebTool vorgestellt wurde (siehe Abschnitt 3.4.3.4 auf Seite 45).

3.5.2.2 Dynamische Generierung

Um einzelne Webseiten nach Bedarf und somit nicht stets den gesamten Webauftritt zu generieren, ist auch ein dynamischer Modus für die HTML-Generierung vorgesehen. Dann wird die jeweils angeforderte Webseite *ad hoc* generiert und unmittelbar dargestellt. Es sei an dieser Stelle jedoch darauf hingewiesen, dass sich dieser Modus lediglich auf die Erzeugung von HTML-Seiten bezieht und dadurch *nicht* einem dynamischen Einsatzzweck im Sinne einer Webanwendung entspricht.

Wenn von der Funktionalität der dynamischen Seitengenerierung Gebrauch gemacht wird, sollte stets bedacht werden, dass dann die generierten Seiten nirgends, insbesondere nicht auf einem Webserver, persistiert werden. Somit sind sie lediglich "flüchtiger" Natur und entziehen sich deshalb einer Indizierung durch Suchmaschinen. Dieser Modus ist für den weiteren Verlauf dieser Arbeit unbedeutend.

3.5.3 Konfigurationsparameter

Im Zusammenhang mit dem WebTool wurde bereits das dort verwendete Konzept zur Konfiguration des Tools beschrieben (vgl. Abschnitt 3.4.3.4 auf Seite 45 und Abschnitt 3.4.4.3 auf Seite 50). Die Grundidee davon wurde in das jLAF übernommen, dabei jedoch zugleich auch verfeinert. So unterscheidet das Framework explizit zwischen *globalen* und *lokalen* Konfigurationsinformationen. Globale Informationen beziehen sich auf die gesamte Verzeichnisstruktur, die bei der Generierung des Webauftritts berücksichtigt wird, und enthalten beispielsweise Informationen über das Quell- und das Zielverzeichnis sowie insbesondere auch die Angabe, für welche Sprache der aktuelle Generierungsdurchlauf erfolgen soll (siehe dazu auch Abschnitt 3.5.7).

Für jedes Verzeichnis der Ordnerstruktur unterhalb des Quellverzeichnisses existiert zudem noch jeweils eine Datei, mit der lokale – d.h. nur für die Inhalte des aktuellen Verzeichnisses relevante – Konfigurationsoptionen gesetzt werden können.

Um eine möglichst einfache Änderung der Konfigurations-Parameter zu bieten, werden diese in Form einer in Java üblichen *Properties*-Datei vorgehalten[14]. Im weiteren Verlauf dieser Arbeit wird die globale Konfigurationsdatei als *Global Property File (GPF)* und ihr lokales Pendant als *Local Property File (LPF)* bezeichnet.

3.5.3.1 Globale Konfigurationsinformationen (GPF)

Listing 3.3 zeigt ein Beispiel für eine mögliche globale Konfigurationsdatei. Die Zeilen haben dabei die folgende Bedeutung (wobei die Reihenfolge der Angaben keine Rolle spielt):

Listing 3.3: Beispiel für eine globale Konfigurationsdatei.

```
1 fragmentSuffix:_frag
2 templateStdPage:/Users/norbert/myHomepage/templates/template.tmp
3 srcDir:/Users/norbert/myHomepage/
4 destDir:/www/htdocs/
5 configFileName:config.cfg.DE
6 pathToXSLFiles:/Users/norbert/myHomepage/xsl/
7 RAPClass:de.norbertheidenbluth.jlaf.raps.MyRAPs
```

Zeile 1: Inhaltsfragmente werden in Dateien abgelegt, deren Name eine bestimmte Endung besitzen muss. Diese Endung wird hier konfiguriert.

Zeile 2: Soweit (lokal) nicht anders angegeben, kommt bei der Generierung die hier referenzierte Standardvorlage zum Einsatz.

Zeilen 3 und 4: Sie enthalten Angaben zum Quell- und Zielverzeichnis der Dateien.

Zeile 5: Diese Information gibt den Namen der lokalen Konfigurationsdateien (LPF) an. Im gegebenen Beispiel ist zu erkennen, dass es sich dabei um Konfigurationparameter handelt, die zur Generierung des Webauftritts in deutscher Sprache führen werden (Endung `DE`); entsprechend lässt sich durch eine einfache Änderung dieser Eigenschaft das Resultat der Generierung entscheidend beeinflussen.

Zeile 6: Diese Eigenschaft ist nur relevant, falls im Rahmen der Generierung XML-Dokumente verarbeitet werden, die mit Hilfe von Stylesheet-Transformationen in HTML-Seiten übertragen werden sollen. Dann gibt diese Eigenschaft an, in welchem Verzeichnis sich die Stylesheet-Dateien befinden.

Zeile 7: In Abschnitt 3.5.5 wird das Konzept der *Remote Access Points (RAPs)* erläutert. Diese Zeile gibt den Namen der Klasse an, welche die RAPs enthält.

[14] Eine *Properties*-Datei ist eine reine Textdatei, deren Zeilen aus einem Schlüssel-Wertepaar besteht. Der Schlüssel wird beim Auslesen der Datei als Eigenschaftsname (*Property Name*) verwendet, dem der angegebene Wert (oder eine Liste von Werten) zugeordnet wird.

3.5.3.2 Lokale Konfigurationsinformationen (LPF)

Auch die lokalen Konfigurationsinformationen werden in einer `Properties`-Datei vorgehalten und entsprechend als Schlüssel-Werte-Paar eingelesen (hier: mit dem Doppelpunkt als Trennzeichen). Damit auch Listen von Werten angegeben werden können, wird die Bezeichnung vor dem ersten Trennzeichen (Doppelpunkt) als Schlüssel, der Rest der Zeile als Wert bzw. Liste von (abermals jeweils durch einen Doppelpunkt voneinander getrennten) Werten interpretiert. Schlüsseln, deren Bezeichnung innerhalb zweier "`@`"-Zeichen eingebettet ist, kommt eine Sonderbedeutung zu, denn sie werden bei der Generierung als *Direktiven* interpretiert, d.h. sie lösen eine Aktion aus (vgl. Beispiel 3.4).

Listing 3.4: Beispiel für eine lokale Konfigurationsdatei.

```
1 @generate@:impressum._frag:Impressum
2 @copyAsIs@:pictures
3 @copyAsIs@:styleSheets
4 @generate@:content._frag:Inhalt
5 @generate@:aboutUs._frag:Ueber uns
6 @generate@:lectures._frag:Vorlesungen
7 @generateDir@:ta:Tutoren
```

Beispiel: Listing 3.4 zeigt ein Beispiel für eine mögliche lokale Konfigurationsdatei, welche ausschliesslich aus Direktiven besteht. Es sind in diesem Beispiel drei Direktiven zu unterscheiden:

Direktive `generate`: Ihr werden zwei Parameter übergeben, woraufhin sie die Generierung des als erster Parameter genannten Fragments auslöst, der sich unter der als zweiter Parameter übergebenen Bezeichnung in die Navigationsstruktur einbringt.

Direktive `generateDir`: Sie entspricht der Direktive `generate`, wobei diesmal nicht eine Fragment-Datei sondern ein komplettes (Unter-)Verzeichnis zur Generierung angegeben wird.

Direktive `copyAsIs`: Sie kopiert die als Parameter übergebene Datei bzw. das Verzeichnis so, wie es ist, in das Zielverzeichnis. Eine Generierung wird dabei nicht vorgenommen. Wenn ein Verzeichnis angegeben wurde, hat dies zur Folge, dass auch dessen Inhalte lediglich kopiert werden und von der Generierung ausgenommen bleiben.

Das Direktiven-Konzept wird in Abschnitt 3.5.5 nochmals aufgegriffen.

3.5.4 Erläuterungen zu ausgewählten Paketen des Frameworks

Die Klassen und Interfaces des Frameworks sind in einzelnen Paketen organisiert. Einzelne, konzeptionell besonders bedeutsame Pakete bzw. Klassen und Interfaces werden im Folgenden näher erläutert. Abbildung 3.8 gibt zudem einen abstrakten Überblick über den Zusammenhang der wichtigsten Klassen in Form eines UML-Paketdiagramms. Für die nachfolgenden Erläuterungen nicht wesentliche Pakete (z.B. eigene Exceptions) wurden aus Gründen der besseren Übersichtlichkeit in Abbildung 3.8 nicht aufgenommen.

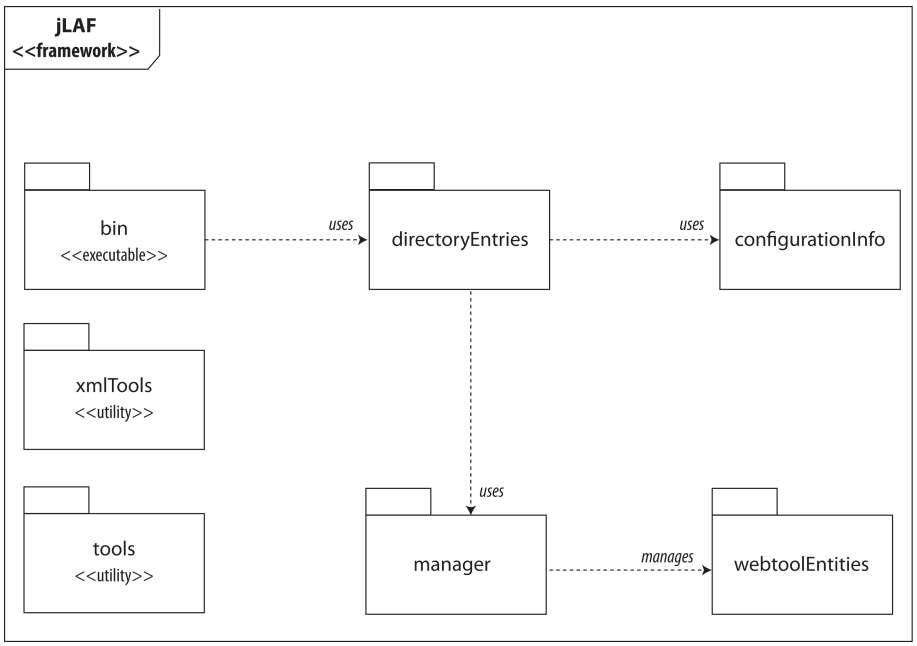


Abbildung 3.8: Gesamtzusammenhang der wichtigsten Pakete des jLAF-Frameworks

3.5.4.1 Die `WebtoolEntity`-Hierarchie

Das jLAF-Framework unterstützt einen einfachen Umgang mit XML-Daten dadurch, dass es mit der abstrakten Klasse `WebtoolEntity` einen *hot spot*[15] für eigene *Data Access Objects (DAO)* bietet. Aus den aus XML-Dateien (oder anderen externen Quellen) eingelesenen Daten werden DAOs instanziiert, um deren Verwaltung sich entsprechende Manager-Klassen (siehe nächster Abschnitt) kümmern. Die jeweiligen DAO-Klassen sind Implementierungen der abstrakten Oberklasse `WebtoolEntity` und müssen vom Benutzer erstellt werden. Im weiteren Verlauf dieser Arbeit wird immer wieder auf den Begriff einer (Webtool)-*Entity* Bezug genommen. Dann ist stets eine konkrete DAO-Instanz gemeint. Entsprechend bezeichne im folgenden der Terminus *Entity-Klasse* die implementierende Klasse (also nicht eine konkrete Instanz). Die Bedeutung der `WebtoolEntity`-Hierarchie wird in Kapitel 5 von Bedeutung sein und dort nochmals aufgegriffen.

3.5.4.2 Die abstrakte Klasse `WebtoolEntityManager`

In der üblichen Verwendung des Frameworks werden aus den Klassen der WebtoolEntity-Hierarchie jeweils viele Instanzen erzeugt und zu einer Einheit (in Form von Container-Klassen, im folgenden auch `Collection` genannt) zusammengefasst. Ferner ist – wie im vorangegangenen Unterabschnitt angedeutet – eine Datenquelle (z.B. eine Datei) der Lieferant für die Informationen, aus denen dann Instanzen der Entity-Klassen erzeugt werden.

Zur bequemen Erzeugung der Entity-Collections und insbesondere auch zu deren Verwaltung sieht das Framework eine weitere Klassenhierarchie vor, deren abstrakte Oberklasse der `WebtoolEntityManager` ist. Zu jeder konkreten Entity-Klasse sollte eine zugehörige Manager-Klasse existieren, die sich um die Collections "ihrer" Entity-Instanzen kümmert.

[15] Der Begriff *hot spot* bezeichnet eine jener Stellen im Framework, an der das Framework vom Benutzer den individuellen Anforderungen und Gegebenheiten angepasst werden kann.

Die wichtigste Methode, die in der Klasse `WebtoolEntityManager` abstrakt definiert ist und deshalb einer Implementierung in den konkreten Unterklassen bedarf, ist die in Code-Ausschnitt 3.5 genannte Methode `buildEntities()`. Sie hat die Aufgabe, die Collections (hier: Vektoren) der entsprechenden Entity-Objekte zu erzeugen, wobei die Verantwortung für das "Wie" und "Woher" den jeweiligen konkreten Implementierungen vorbehalten bleibt. So kann eine Implementierung beispielsweise eine Datenbank kontaktieren, um die zur Erzeugung der Entity-Objekte notwendigen Informationen zu erhalten, eine andere jedoch diese Informationen aus Dateien beziehen, wohingegen eine dritte Implementierung einen *Remoting*-Mechanismus einsetzt. Fehler beim Instanziieren der DAOs werden durch das Werfen einer Exception bekannt gemacht.

Listing 3.5: Die abstrakte Methode `buildEntites` der Klasse `WebtoolEntityManager`

```
1 public abstract Vector<? extends WebtoolEntity> buildEntities()
2    throws WebtoolException;
```

Die Verwendung der Entity-Klassen und der zugehörigen Verwalter-Klassen (*Manager*) in der Praxis sowie ihre Bedeutung (auch) für die Generierung von HTML-Seiten wird im Praxisbeispiel (Abschnitt 5.5 auf Seite 135) erläutert.

3.5.4.3 Das Paket `configurationInfo`

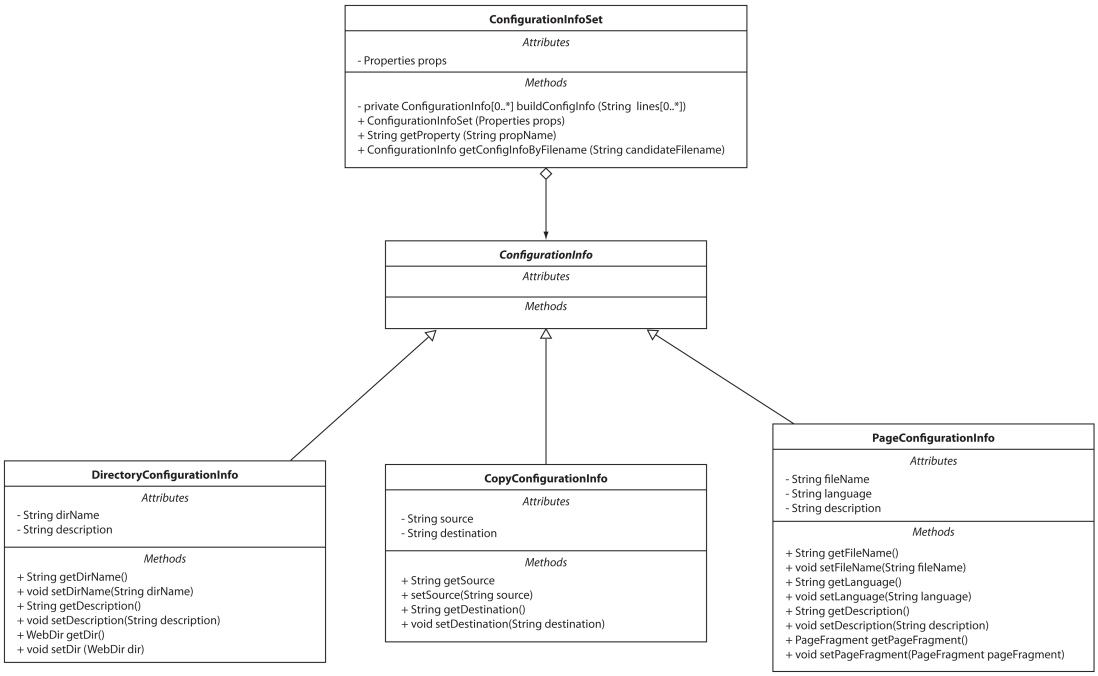


Abbildung 3.9: Klassenhierarchie im Paket `configurationInfo`

Die interne Handhabung der globalen Konfigurationsinformationen (GPF, vgl. Abschnitt 3.5.3) ist recht einfach durch eine entsprechende Instanz der Klasse `Properties` aus einer Java-Standard-Bibliothek zu realisieren, denn die Namen der Eigenschaften sind im voraus bereits bekannt, sodass letztendlich nur das Zuordnen der jeweiligen Schlüssel zur Laufzeit von Bedeutung ist. Lokale Informationen (LPF) hingegen sind vor Beginn der Generierung *nicht* bekannt, denn jedes Verzeichnis kann (und wird) sich inhaltlich von allen anderen Verzeichnissen

unterscheiden. Infolgedessen müssen für jedes Verzeichnis separat die lokalen Konfigurationsparameter ausgelesen und verwaltet werden.

Aufgrund seiner objektorientierten Ausrichtung erlaubt das jLAF einen eleganten Umgang mit den lokalen Konfigurationsinformationen: um verschiedene Arten von Konfigurationsinformationen zu verwalten, stehen im Paket `configurationInfo` entsprechende Klassen bereit, die jeweils Implementierungen der abstrakten (Ober-)Klasse `ConfigurationInfo` sind. Das Framework bringt bereits drei Implementierungen mit:

Klasse `DirectoryConfigurationInfo`**:** Sie dient zur Verwaltung von Konfigurationsinformationen für eine komplettes (Unter-)Verzeichnis.

Klasse `PageConfigurationInfo`**:** Sie dient zur Verwaltung von Konfigurationsinformationen für eine einzelne Webseite.

Klasse `CopyConfigurationInfo`**:** Sie enthält Informationen zum Kopieren von Dateien oder (Unter-)Verzeichnissen, beispielsweise Quell- und Zielpfade. Diese Informationen sind dann relevant, wenn Inhalte eines Verzeichnisses nicht in die Generierung einbezogen, sondern lediglich so, wie sie sind, kopiert werden sollen.

Instanzen der genannten Klassen werden beim Auslesen der LPF zur Laufzeit erzeugt und in einen eigenen, typisierten Container aufgenommen. Als solcher dient die ebenfalls im Paket enthaltene Klasse `ConfigurationInfoSet`. Dabei ist sichergestellt, dass die Reihenfolge, in der die Einträge in der Konfigurationsdatei stehen, in der Containerklasse erhalten bleibt. Diese Tatsache ist durchaus von Bedeutung, denn die Konfigurationsinformationen werden nicht nur bei der Generierung der Seiten, sondern auch bei der Erzeugung der Navigationsstrukturen verwendet. Das bedeutet, dass die Reihenfolge der Einträge in den Navigationsstrukturen nicht zufällig oder von anderen Faktoren abhängig ist (siehe hierzu auch [Hei04a]), sondern präzise durch die Positionen der Einträge in den lokalen Konfigurationsdateien gesteuert werden kann. Instanzen der Klassen `DirectoryConfigurationInfo` und `PageConfigurationInfo` besitzen jeweils die Eigenschaft `description`. Diese entspricht dem Text, unter welchem sich der jeweilige Eintrag in die Navigationsstruktur(en) einbringen soll. (Eine Mehrfachnennung ist auch möglich.)

3.5.4.4 Das Paket `directoryEntries`

Die im Paket `directoryEntries` (vgl. Klassendiagramm in Abbildung 3.10) zusammengefassten Klassen repräsentieren Inhalte eines Verzeichnisses im Dateisystem. Grundlage dieses Pakets ist das Interface `IDirectoryEntry`[16], durch dessen Implementierung eine Methode zum Schreiben des generierten Inhalts in eine Ausgabedatei bereitgestellt wird. Implementierende Klassen dieses Interfaces sind (mindestens) die Klassen `WebDir` und `PageFragment`:

Die Klasse `WebDir`**:** Instanzen dieser Klasse repräsentieren ein komplettes Verzeichnis. Dementsprechend besteht ihre Aufgabe primär in der Verwaltung eine Menge weiterer Objekte einer Klasse des Pakets `directoryEntries`, d.h. Instanzen der Klassen `WebDir` oder `PageFragment` (bzw. deren Ableitungen). Auf diese Weise lässt sich zugleich auch die

[16] Konvention: die Benennungen von Interfaces beginnen im Rahmen dieser Arbeit stets mit dem Buchstaben "I", um somit eine einfache Unterscheidung zwischen Klassen und Interfaces bereits aufgrund deren Bezeichnungen zu erreichen.

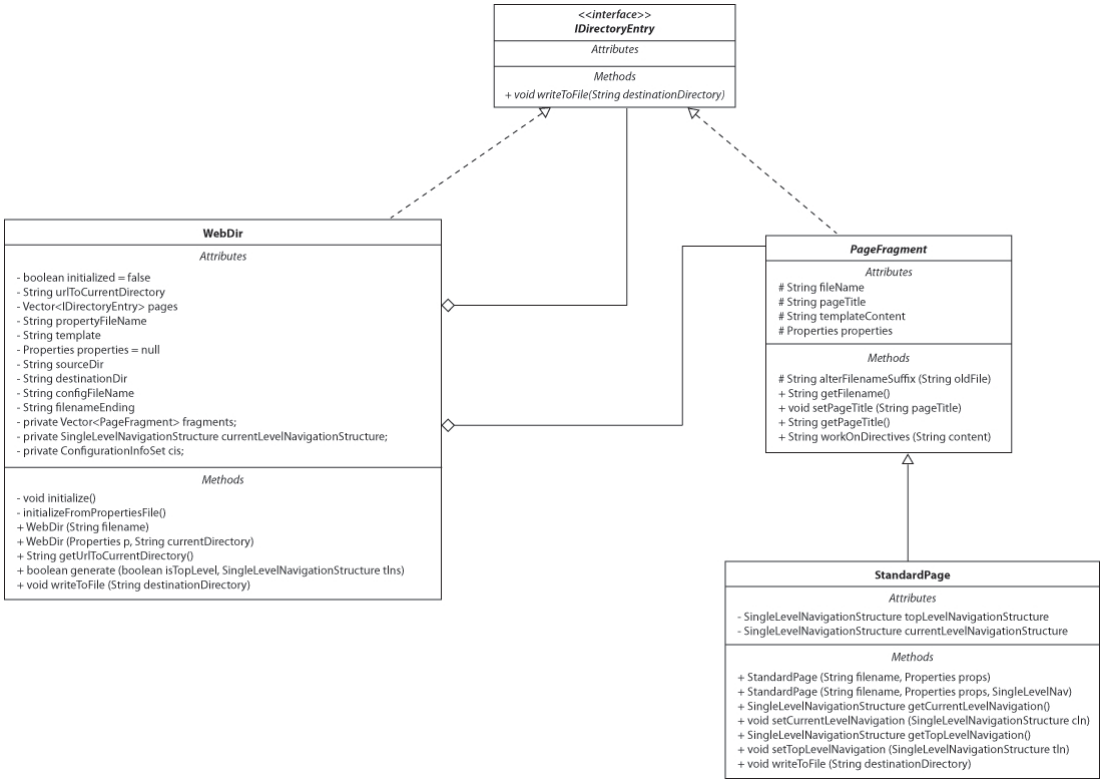


Abbildung 3.10: Klassenhierarchie im Paket `directoryEntries`

Hierarchie des gesamten Webauftritts modellieren: die "Vaterinstanz" der Klasse `WebDir` erstellt und referenziert für jedes Unterverzeichnis des aktuellen Verzeichnisses genau eine weitere Instanz dieser Klasse, welche auf ihrer Verzeichnisebene wiederum genauso verfährt und damit als Vaterinstanz für ihre eigenen Unterverzeichnisse fungiert. Der Aufruf der Instanzmethode `generate()` auf der Wurzel-Instanz startet die (rekursive) Generierung des gesamten Webauftritts. Ein Code-Beispiel für einen solchen Aufruf zeigt Listing 3.6.

Die Klasse `PageFragment`: Die Klasse `PageFragment` ist abstrakt und dient als ein weiterer *hot spot* des Frameworks. Implementierende Klassen von `PageFragment` sind Repräsentierungen einer einzelnen Webseite. Sie enthalten Platzhalter für die Stellen, an denen die Seitenelemente wie beispielsweise Kopfzeile, Navigationsstruktur(en), Fußzeile usw. eingesetzt werden. Weil insbesondere die Frage der Seitengestaltung höchst individuell ist und vermutlich von jedem Benutzer des Frameworks unterschiedlich gehandhabt wird, verbleibt ihm (dem Benutzer) die Aufgabe, eine oder mehrere Klasse(n) zu schreiben, die von `PageFragment` erben und insbesondere die im Interface `IDirectoryEntry` geforderte Methode `writeToFile` implementieren. Weil diese Implementierung *nicht* direkt in der Klasse `PageFragment` enthalten ist, handelt es sich dabei um eine abstrakte Klasse. Konkrete Implementierungen der Klasse `PageFragment` können mithin als Vorlage (*Template*) für eine Darstellungsform einer Webseite verstanden werden.

Das Paket `directoryEntries` enthält mit der Klasse `StandardPage` eine Beispielimplementierung der abstrakten Klasse `PageFragment`. Diese repräsentiert eine Webseite, deren Struktur derjenigen in Abbildung 3.1 auf Seite 42 entspricht: es gibt einen für alle Seiten des Webauftritts identischen Kopfzeilenbereich (z.B. für ein Logo), einen pro Seite individuell ausgestalteten In-

haltsbereich sowie einen dritten Bereich zur Aufnahme der Navigationsstruktur. Somit lassen sich mit dem Framework bereits ohne eigene Erweiterungen kleine und mittlere Webauftritte[17] erstellen und verwalten. Informationen und Anweisungen zur *Anordnung* dieser Elemente auf der Seite sind in der Klasse nicht enthalten, weil jedwede Layoutangaben unabhängig von der Struktur der Seite gehalten und über Cascading Style Sheets (CSS) gesteuert werden. Lediglich die *Struktur* der Seite wird durch die jeweilige konkrete Klasse unterhalb von `PageFragment` repräsentiert.

Listing 3.6: Beispiel für das Auslösen der Generierung eines kompletten Webauftritts.

```
1  String propertyFile = "myWebsiteProperties.props";
2  // property file contains path to source and destination directories
3  WebDir rootDir = new WebDir(propertyFile); // represents Website's root dir
4  // start recursive generation; if generation was successful,
5  // generate() returns true, otherwise false
6  if (rootDir.generate()) {
7     System.out.println("Your website has been build successfully.");
8  } else {
9     System.err.println("Generation aborted due to errors.");
10 }
```

Die Ausführungen zum Paket `directoryEntries` sollen mit einem Quellcode-Ausschnitt, der den initialen Aufruf der Generierung eines kompletten Webauftritts exemplarisch zeigt, schließen (siehe Listing 3.6). In diesem Quellcode wird das für Frameworks typische Prinzip des *Inversion of Control (IoC)* erkennbar: das Framework tritt an die Stelle des Hauptprogramms und steuert den Ablauf der Anwendung (hier: die Generierung der Seiten, angestoßen durch den Aufruf der Methode `generate()` in Zeile 6). Die Aufgabe des Programmierers bleibt darauf beschränkt, durch das Implementieren der abstrakten Klassen bzw. Methoden (hier: in der Klasse `PageFragment`) eine individuelle Ausgestaltung der Klassen zur Repräsentierung einzelner Webseiten zu erreichen.

3.5.4.5 Das Paket `xmlTools`

Der objektorientierte Ansatz sowie der Charakter des Frameworks erlauben eine nahtlose Integration der bereits im Konzept des WebTools bedeutsamen Ideen zur XML-Transformation: an die Stelle des externen Tools `xmlstarlet`, das im WebTool die Validierung und Transformation von XML-Dokumenten vornahm, tritt nun das im Framework enthaltene Paket `xmlTools`. Es enthält eine Klasse zur Stylesheet-Transformation von XML-Dokumenten sowie eine abstrakte Klasse, deren Implementierungen auf das Auslesen und Verarbeiten jeweils bestimmter XML-Dokumente spezialisiert sind.

Die Klasse `XSLT-Transformer`: Instanzen dieser Klasse dienen als reine Werkzeuge (d.h. sie haben keinen unmittelbaren Bezug zum Framework) und nehmen die Transformation eines XML-Dokuments (angegeben entweder direkt oder als URL zur XML-Datei) mit Hilfe eines XML-Stylesheets (angegeben durch eine URL) in einen Ausgabekanal (*output stream*) vor.

Die Klasse `AbstractXMLParser`: Diese Klasse ist abstrakt und dient als *hot spot* für individuelle Anpassungen des Frameworks. Implementierungen dieser Klasse sind darauf spezialisiert, jeweils eine bestimmte Art von XML-Dokumenten einzulesen und aus den darin

[17] für eine Erklärung zu den Attributen "klein" und "mittel": siehe Abschnitt 3.4.1.1 auf Seite 39

enthaltenen Informationen eine Datenstruktur (hier: Liste) von zugehörigen Objekten aufzubauen. Zum Erhalten dieser Liste steht die öffentliche Methode `public List<WebtoolEntity> getElements(String elementName)` zur Verfügung. Sie bekommt als Parameter den Namen des XML-Tags übergeben, aus dessen jeweiligen Vorkommen im XML-Dokument Objekte erzeugt werden sollen, und liefert eine Liste dieser Objekte zurück, welche für die Verwendung mit konkreten Implementierungen der abstrakten Klasse `WebtoolEntity` typisiert ist. An dieser Stelle greift auch die Validierung der Daten gegen eine DTD- bzw. Schema-Datei, welche bereits im Zusammenhang mit dem WebTool erläutert wurde (vgl. Abschnitt 3.4.5 auf Seite 53). Während beim WebTool zu diesem Zweck das Tool `xmlstarlet` verwendet wurde, steht in Java mit dem Paket `javax.xml.validation` für die Validierung eine eigene Programmierschnittstelle zur Verfügung. Die prinzipielle Funktionsweise der Validierung ist in jLAF jedoch identisch mit derjenigen im WebTool.

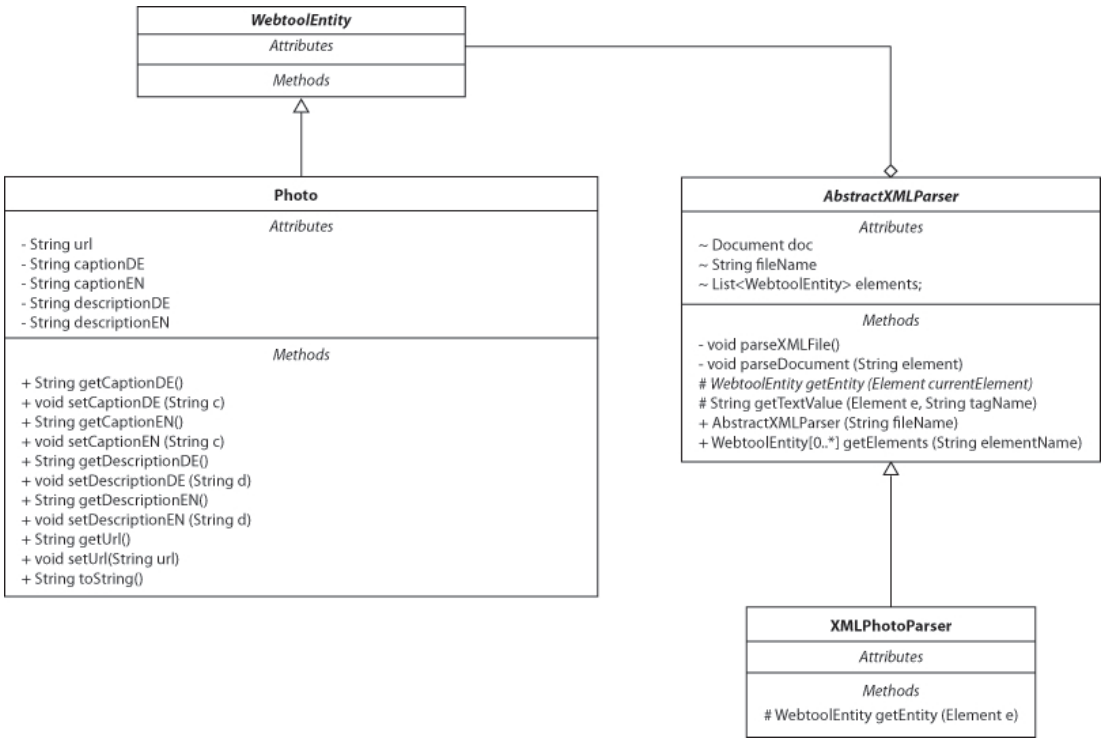


Abbildung 3.11: Abstrakte Klassen und deren individuelle Implementierungen zum Umgang mit XML-Dokumenten

Beispiel: Die abstrakte Klasse `AbstractXMLParser` bietet eine Methode an, welche aus einem XML-Dokument eine Liste von `WebtoolEntity`-Objekten erzeugt. Beispielhaft zeigt Abbildung 3.11 das entstehende Klassendiagramm, wenn zur Laufzeit zwei konkrete Implementierungen der genannten abstrakten Klassen verwendet werden. In der Abbildung sind dies die Klassen `XMLPhotoParser` bzw. `Photo` (welche zum Umgang mit foto-bezogenen Daten bzw. entsprechenden XML-Dokumenten dienen). Wie Abbildung 3.11 zu entnehmen ist, erwartet die abstrakte Klasse `WebtoolEntity` keine Implementierungen von Methoden. Sie dient vielmehr lediglich dazu, eine *is-a*-Beziehung der von ihr erbenden Klassen zum Ausdruck zu bringen. Die abstrakte Klasse `AbstractXMLParser` hingegen erwartet die Implementierung der Methode `WebtoolEntity getEntity(Element currentElement)`, welche aus einem als Parameter übergebenen Elementknoten des XML-Dokuments ein Objekt erstellt.

Listing 3.7: XML-Datei zur Verwaltung von Foto-Informationen

```
<myPhotos>
   <image>
      <url>/Users/norbert/Pictures/pic0001.jpg</url>
      <captionDE>Campus der Universitaet Ulm</captionDE>
      <captionEN>Campus Ulm University</captionEN>
      <descriptionDE>Blick auf den Nordeingang</descriptionDE>
      <descriptionEN>View onto northern entry</descriptionEN>
   </image>
   <image>
      ...
   </image>
</myPhotos>
```

Liegen die Implementierungen der abstrakten Klasse vor, so ergibt sich ein bequemer Aufruf zur Erzeugung der `WebtoolEntity`-Kollektionen. Ein Beispiel für einen solchen Aufruf zeigt Listing 3.8. In Zeile 1 wird die konkrete Implementierung `XMLPhotoParser` instanziiert (sie bekommt als Parameter den vollständigen Pfad zur XML-Datei übergeben). Zeile 2 initiiert den Aufbau der Kollektion durch den Aufruf der Methode `getElements`, welche als Parameter die Information erhält, in welchem Tag das entsprechende Objekt in der XML-Datei eingeschlossen ist. Im Beispiel ist dies das Tag `image` (siehe dazu auch die zugehörige XML-Datei in Listing 3.7). Die übrigen Zeilen des Listings 3.8 dienen nur zur Illustration, wie mit der erhaltenen Kollektion weiter gearbeitet werden kann.

Listing 3.8: Verwendung eines konkreten XML-Parsers

```
1 XMLPhotoParser pp = new XMLPhotoParser("/Users/norbert/data/photos.xml");
2 List<WebtoolEntity> entList = pp.getElements("image");
3 for (WebtoolEntity ent: entList) {
4   Photo p = (Photo) ent;
5   // do something with this entity
6 }
```

3.5.5 Das Direktivenkonzept: Dynamische Methodenaufrufe via Reflection

Seitenfragmente, aus denen vollständige Webseiten generiert werden, können aus reinem HTML-Code bestehen. In diesem Fall erfolgt die Generierung wie bereits im Zusammenhang mit dem WebTool beschrieben: sie werden in die Vorlage so, wie sie sind, eingebettet.

Das Framework sieht aber noch eine leistungsfähigere Form der Seitengenerierung vor: Seitenfragmente werden während der Generierung daraufhin untersucht, ob sie *Direktiven* enthalten. Wie bereits zuvor bei den lokalen Konfigurationsdateien beschrieben (vgl. Abschnitt 3.5.3), handelt es sich bei Direktiven um einen Text, der zwischen zwei "`@`"-Zeichen eingebettet ist. Konkret entspricht dieser Text einem Methodennamen, der – optional – um eine Parameterliste ergänzt werden kann. Wann immer bei der Generierung eine Direktive gefunden wird, erfolgt ein Aufruf der angebenen Methode (ggf. nebst Parameterliste). Die Information, in welcher Klasse und in welchem Paket sich die aufzurufenden Methoden befinden, ergibt sich dabei aus einem Eintrag in der globalen Konfigurationsdatei (vgl. Zeile 7 in Listing 3.3 auf Seite 64). Als Rückgabetyp der aufgerufenen Methoden wird ein String erwartet, der dann im generierten Dokument an die Stelle der Direktive platziert wird.

Listing 3.9: Ein komplettes Seitenfragment mit Direktiven

```
1 <h2>Forschung und Lehre</h2>
2 <p>Auf dieser Seite finden Sie Informationen &uuml;ber meine Aktivit&auml;ten in
      Forschung und Lehre.</p>
3 <h4>Meine Publikationen</h4>
4 @transformXMLFromFile(/Users/norbert/data/myPublications.xml,/Users/norbert/
    dummy/webtoolDummy/xsl/publications.xsl)@
5 <h4>Lehrveranstaltungen</h4>
6 @getLecturesInHTML(/Users/norbert/myLectures.txt)@
```

Beispiel: Das Konzept der Direktiven demonstriert Listing 3.9. Die Zeilen 1 bis 3 sowie Zeile 5 enthalten gewöhnlichen HTML-Code. In den Zeilen 4 und 6 hingegen befinden sich Direktiven. Das heißt, an ihrer Stelle befindet sich in der generierten Seite der Rückgabewert der Methoden (welcher deshalb vom Typ `String` sein muss). So wird in Zeile 4 eine Methode zur Stylesheet-Transformation einer XML-Datei aufgerufen (die Namen der XML-Datei sowie der Stylesheet-Datei werden dabei als Parameter übergeben). Zeile 6 erzeugt die HTML-Repräsentation einer Aufzählung (hier: von Vorlesungsinformationen), deren Inhalt aus der als Parameter übergebenen Datei gewonnen wird.

Selbstverständlich können *alle* Methoden potentiell als Direktive verwendet werden (sofern sie einen Rückgabetyp `String` besitzen). Im Hinblick auf den weiteren Verlauf dieser Arbeit werden aber nur einzelne Methoden für den Einsatz als Direktiven vorgesehen und insoweit in einer eigenen, öffentlich zugänglichen Klasse zusammengefasst. In der im Framework verwendeten Terminologie werden derartige Methoden als *Remote Access Points (RAP)* bezeichnet. Die Benennung soll zum Ausdruck bringen, dass RAPs explizite Einstiegspunkte darstellen, um "von außen" auf Funktionalitäten zugreifen zu können, wobei "von außen" mehrere Bedeutungen haben kann. Aktuell ist damit der Aufruf während der Generierung gemeint, später folgen weitere Verwendungsszenarien. Es sei darauf hingewiesen, dass die Verwendung expliziter RAPs für die in diesem Abschnitt gegenständliche Generierung von HTML-Seiten nicht erforderlich ist. Aufgrund seiner Bedeutung im weiteren Verlauf dieser Arbeit (vgl. Abschnitt 5.2.1 auf Seite 111 und Abschnitt 5.3.3.2 auf Seite 123) wird sie jedoch empfohlen.

Bereits in der bislang erläuterten Form steht mit dem Direktiven-Konzept ein sehr mächtiges Instrument zur Steuerung der Generierung zur Verfügung. Es ermöglicht das Einbetten von Inhalten, die erst bei der Generierung aus externen Quellen geholt werden und zu Ausgabe-Seiten zusammen-"gebaut" werden (*"Content Assembling"*).

Die technische Realisierung des Methodenaufrufs auf erst zur Laufzeit bekannten RAPs erfolgt durch die in Java integrierte *Reflection-API*. Jedoch stellt der Einsatz von jLAF keine Anforderungen an die Kenntnisse dieser Programmierschnittstelle, denn das Framework kapselt den Reflection-Mechanismus durch eigene, leicht zu handhabende Methoden. Zu diesem Zweck stellt das Framework die Klassen `ReflectionHandler` und `RemoteMethod` zur Verfügung (vgl. Abbildung 3.12).

Instanzen der Klasse `RemoteMethod` repräsentieren genau eine RAP-Methode und können zudem eine Liste derjenigen Parameter besitzen, die dieser Methode beim Aufruf übergeben werden sollen. Instanzen dieser Klasse werden zur Laufzeit immer dann erzeugt, wenn eine Direktive gefunden wird. Der tatsächliche Aufruf der Methoden wird durch die Klasse `ReflectionHandler` vorgenommen: sie stellt einen Konstruktor zur Verfügung, der ein Objekt der Klasse `RemoteMethod` als Parameter erwartet. Ein Aufruf von `process()` führt dann zum Aufruf der gewünschten Methode. Listing 3.10 zeigt die interne Verarbeitung von Direktiven.

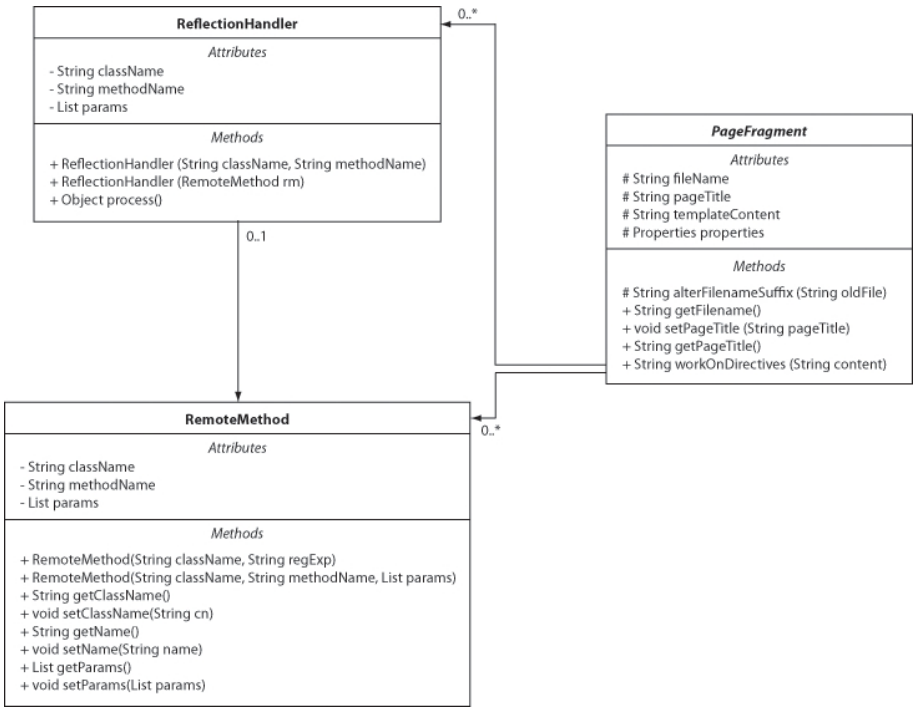


Abbildung 3.12: Die Klassen `ReflectionHandler` und `RemoteMethod`

Listing 3.10: Interne Verarbeitung von Direktiven

```
1  public String workOnDirectives(String content) throws WebToolException {
2    Pattern pat = Pattern.compile("@(.+)@"); // find next directive in content
3    Matcher mOut = pat.matcher(content);
4    while (mOut.find(0)) {
5      RemoteMethod rm = new RemoteMethod(
6        properties.getProperty("RAPClass"), mOut.group(1));
7      ReflectionHandler rh = new ReflectionHandler(rm);
8      content = mOut.replaceFirst((String) rh.process());
9      mOut.reset(content);
10   }
11 }
```

Erklärungen zu Listing 3.10: Die Methode `workOnDirectives` dient dazu, eine Direktive aufzurufen. Als (einziger) Parameter wird ihr der gesamte Inhalt eines eingelesenen Seitenfragments übergeben. Unter Verwendung von regulären Ausdrücken wird der Inhalt daraufhin untersucht, ob er mindestens eine Direktive enthält (Zeilen 2 bis 4). Falls ja, so wird in den Zeilen 5 und 6 eine Instanz der Klasse `RemoteMethod` erzeugt. Hier ist zu sehen, dass die Information darüber, durch welche Klasse die RAPs zur Verfügung gestellt werden, als Parameter übergeben werden kann (im Codebeispiel wird diese Information aus der globalen Konfiguration geholt). In Zeile 7 wird ein `ReflectionHandler` instanziiert, und Zeile 8 enthält den entscheidenden Aufruf der Methode `process()`, welcher letztlich dem Aufruf der in der Direktive genannten Methode entspricht. Die Notation in Zeile 8 führt dazu, dass die betreffende Direktive in dem als Parameter übergebenen String `content` durch den Rückgabewert der aufgerufenen RAP-Methode ersetzt wird. (Wie weiter oben beschrieben, müssen RAP-Methoden aus diesem Grund stets den Rückgabetyp `String` haben.) Das beschriebene Vorgehen wiederholt sich aufgrund der `while`-Schleife in den Zeilen 4 bis 10).für jede gefundene Direktive, die

in dem übergebenen Inhaltsfragment enthalten ist.

Es sei an dieser Stelle noch einmal betont, dass die Verwendung der Klassen `ReflectionHandler` und `RemoteMethod` lediglich als Kapselung der Reflection-API dient und dadurch eine stark vereinfachte Nutzung des Reflection-Mechanismus ermöglicht. Insoweit kann das Direktiven-Konzept auch ohne die Klassen `ReflectionHandler` und `RemoteMethod` verwendet werden, indem stattdessen direkt gegen die Reflection-API programmiert wird.

3.5.6 Berücksichtigung der Barrierefreiheit im jLAF

Da das jLAF konzeptionell dem WebTool entspricht, gelten die Ideen zur Realisierung der Barrierefreiheit analog. Insbesondere setzt auch das jLAF auf eine kontinuierliche Validierung der generierten Inhalte gegen etablierte Standards (vgl. Abschnitte 3.4.3.6 und 3.4.5).

Die Kombination des Direktiven-Konzepts mit den Konfigurations-Parametern ergibt eine sehr einfach zu handhabende Lösung für das Erstellen von auf bestimmte Bedürfnisse angepassten Variationen des Webauftritts. Allein die Änderung des in der globalen Konfigurationsdatei gesetzten Wertes für den Namen der lokalen Konfigurationsdateien (Zeile 5 in Listing 3.3) kann eine völlig andere Ausgabe zur Folge haben. Ebenso ist eine gänzlich andere Fassung möglich, wenn in Zeile 7 eine andere Klasse für die RAPs angegeben wird. Insbesondere birgt eine Kombination dieser beiden Änderungen ein enormes Potenzial für die Individualisierbarkeit des erzeugten Webauftritts.

Ein Beispiel für das auf diese Weise vorhandene Individualisierungs-Potential liegt in Form der im nächsten Unterabschnitt erläuterten Mehrsprachigkeit vor. Ebenso gut kann auf diese Weise aber auch eine Variante generiert werden, die anstelle von Grafiken textuelle Repräsentationen beinhaltet oder in anderweitiger Form auf spezielle Bedürfnisse zugeschnitten ist. Speziell auf einen Ausgabe-Kontext zugeschnittene RAPs in Verbindung mit ebenfalls auf diesen Kontext ausgerichteten lokalen Konfigurationsinformationen lassen sich durch simple Änderungen der Zeilen 5 bzw. 7 in Listing 3.3 auf Seite 64 aktivieren.

3.5.7 Mehrsprachigkeit

Eine Anforderung an das WebTool war die Unterstützung der Mehrsprachigkeit (Merkmal 1d auf Seite 40). Die Realisierung dieser Anforderung unterscheidet sich im jLAF von derjenigen im WebTool, sodass an dieser Stelle darauf erneut eingegangen werden soll.

Das WebTool erkennt anhand der Dateinamen-Appendix, um welche Sprache es sich bei der aktuell zu verarbeitenden Datei handelt (siehe Seite 47f.). Im jLAF hingegen erfolgt jede Generierung explizit für eine spezielle Sprache. Mit Hilfe der globalen Konfigurationsinformationen (GPF, siehe Abschnitt 3.5.3) wird diese Sprache vor Beginn der Generierung festgesetzt. Aufgrund dieser globalen Angabe werden auf lokaler Ebene dann die jeweils passenden lokalen Konfigurationsdateien (LPF) bei der Generierung verwendet. Mit anderen Worten: für jede verfügbare Sprache existieren eigene lokale Konfigurationsdateien. Der Vorteil dieses Ansatzes (im Unterschied zum WebTool) liegt darin, dass auf diese Weise auch die entfernten Klassen (RAPs, vgl. Abschnitt 3.5.5) sprachabhängig aufgerufen werden können: einerseits kann ihr Aufruf mit für jede Sprache unterschiedlichen Parametern erfolgen, andererseits ist es aber auch denkbar, dass für unterschiedliche Sprachen auch unterschiedliche RAPs angesprochen werden. Das jLAF erlaubt mithin also eine deutlich detailliertere Lokalisierung als das WebTool.

3.5.8 Diskussion des jLAF

Mit dem jLAF liegt ein Framework vor, das die Konzepte und Ziele des WebTools in einen objektorientierten, wiederverwendbaren und in Web-Umgebungen integrierbaren Kontext bringt. Es bietet eine präzise Konfigurierbarkeit in globaler und lokaler Hinsicht und kann aufgrund des Direktiven-Konzepts als ein "Orchestrator" betrachtet werden, der aus verschiedensten Quellen Inhalte aggregiert und daraus einen Webauftritt generiert.

3.5.8.1 Abgrenzung zu JavaServerPages

Die Verwendung von Direktiven und Seitenfragmenten erinnert auf den ersten Blick an das Konzept hinter den *Java Server Pages (JSPs)*. Auch in diesem "mischen" sich Aufrufe von Java-Methoden unter den HTML-Code einer Seite und ersetzen diesen Code durch die Rückgaben der aufgerufenen Methoden. Sogar der Terminus "Direktive" wird im Zusammenhang mit JSPs verwendet, wenngleich darunter auch etwas anderes zu verstehen ist als im jLAF. Jedoch unterscheiden sich JSPs bei näherer Betrachtung aber insbesondere deshalb vom Direktiven-Konzept, weil

- Webseiten, die auf JSPs beruhen, stets in einem Applikationsserver laufen müssen und insoweit ausschließlich dynamischer Natur sind. jLAF hingegen bietet die Möglichkeit, die Seiten sowohl für einen statischen Auftritt als auch rein dynamisch zu generieren. Im Falle statischer Seiten reicht deshalb ein Webserver zur Veröffentlichung aus. Ferner unterscheiden sie sich, weil
- JSPs Logik enthalten (können), in den Seitenfragmenten jedoch lediglich Methodennamen nebst eventuellen Parametern notiert werden. Diese führt das Framework dann bei der Generierung der Seiten aus.

Der angedachte Einsatzzweck von JSPs ist überdies ein anderer als bei jLAF. jLAF adressiert – in der bislang vorgestellten Form – die Generierung statischer HTML-Seiten und wird in Kapitel 5 in eine andere Richtung weiterentwickelt. JSPs verstehen sich indes als eine Art "Vermittler" zwischen der Präsentation einer Webseite und der (serverseitigen) Logik, mit Hilfe derer die Seiteninhalte erstellt werden. Der dynamische Einsatz steht hier also im Vordergrund. Mit Ausnahme der Direktive(n) enthält ein Seitenfragment in jLAF keinen weiteren HTML-fremden Code. Die Notation von JSPs weicht die strikte Trennung von HTML und Logik jedoch wieder auf, obwohl genau diese nach [Ber04] eine wesentliche Motivation für den Einsatz von JSPs sein soll. Auf diese Quelle sei überdies für weiterführende Erläuterungen zu JSPs verwiesen.

3.5.8.2 Gemeinsamkeit mit *Plug-In*-Systemen

Bei einer sehr abstrakten Sichtweise auf das jLAF können die in die Navigationsstrukturen aufzunehmenden Einträge (d.h. die Verzeichnisse und Dateien bzw. deren Informationen in den Konfigurationsdateien) als *Plug-Ins* und das jLAF als das zugehörige Plug-In-System angesehen werden. Insofern lassen sich zur Erzeugung der Navigationsstrukturen bei sehr komplexen Hierarchien auch die in [BBHM06a] und [BBHM06b] vorgestellten Lösungsansätze zur Erzeugung der Navigationsstrukturen adaptieren. Sie wurden jedoch bewusst *nicht* zu einem integralen Bestandteil des jLAFs gemacht, weil sich ihr Einsatz erst bei sehr komplexen Strukturen lohnen würde. Das WebTool und jLAF adressieren jedoch – wie zu Beginn des Kapitels

erwähnt – kleine und mittlere Webauftritte, die sich gemäß der Definitionen auf Seite 39 gerade dadurch auszeichnen, nur über schlanke Hierarchien zu verfügen. Im Interesse einer übersichtlichen Gestaltung des jLAF wurde deshalb auf die Adaption und Integration der Ideen aus [BBHM06a] und [BBHM06b] verzichtet.

3.5.8.3 Weiterentwicklung von jLAF

In der bislang beschrieben Ausprägung stellt das Framework im Wesentlichen ein optimiertes Java-Pendant zum Perl-basierten WebTool dar. Deshalb gelten die auf Seite 58 getroffenen Aussagen hinsichtlich der Erfüllung der ATAG-Richtlinien auch sinngemäß für das Framework. Somit erfüllt es bereits in der vorliegenden Form seinen Zweck. Die eigentliche Intention des Frameworks reicht aber weit über den mit dem WebTool beabsichtigten Zweck hinaus: es hilft bei der Generierung von (barrierefreien) Fallback-Lösungen einer bestimmten Klasse von Rich Internet Applications (RIAs). Oder in einer anderen Formulierung: die hier betrachtete Generierung der HTML-Seiten ist ein (erwünschtes!) “Nebenprodukt” beim Einsatz des Frameworks in einem anderen Kontext. (Insbesondere dadurch setzt es sich deutlich von JSPs ab.) Nach einem Überblick über die RIA-Technologien im nachfolgenden Kapitel wird Kapitel 5 dann die entsprechende Erweiterung des Frameworks zum Inhalt haben und den hier angedeuteten “anderen Kontext” präzisieren.

Kapitel 4

Rich Internet Applications

"The power of the Web is in its universality.
Access by everyone regardless of disability is an essential aspect."
(Tim Berners-Lee)

4.1 Motivation

Die bislang in dieser Arbeit betrachteten Webauftritte waren allesamt statischer Natur. Die Übermittlung von Informationen, nicht jedoch eine Interaktion mit dem Besucher der Seiten, stand dabei im Vordergrund, d.h. der Besucher war Konsument der bereitgestellten Informationen, den dargestellten Inhalt beeinflusste er jedoch nicht. Für diese Kategorie von Seiten soll zunächst ein eigener Begriff definiert werden:

Definition: Inhaltszentrierte Webseiten

> Eine Webseite wird als inhaltszentriert bezeichnet, wenn sie auf die reine Vermittlung von Informationen ausgerichtet ist und – mit Ausnahme von Links zu anderen Seiten – keine Interaktionselemente anbietet.

Von inhaltszentrierten Seiten zu unterscheiden sind solche Seiten, welche die Mitwirkung durch den Benutzer in den Mittelpunkt stellen. Ein einfaches Beispiel dafür ist eine Seite, die ein Bestellformular enthält. Auch für diese Kategorie soll eine entsprechende Begriffsdefinition den weiteren Ausführungen vorangestellt werden:

Definition: Interaktionszentrierte Webseiten

> Webseiten werden dann als interaktionszentriert bezeichnet, wenn der Benutzer durch sein Handeln (insbesondere in Form von Eingaben) aktiv den Inhalt und/oder den Ablauf der weiteren Darstellung beeinflusst.

Je nach Ausgestaltung solch interaktionszentrierter Seiten erlebt der Benutzer sie nicht mehr als typische Webseiten, sondern als eine (im Browser laufende) Anwendung: als eine *Internet Application*.

Sowohl für inhalts- als auch für interaktionszentrierte Seiten stellen die Limitierungen von HTML und CSS einen Nachteil dar: die Möglichkeiten bei der Gestaltung der Benutzerschnittstelle sind eingeschränkt und erlauben bei weitem nicht solch flexible Layouts, wie sie von traditionellen Desktop-Applikationen her bekannt sind. Das Öffnen neuer Fenster (sog. *Pop-Ups bzw. PopUp-Windows*) und ähnliche Variationen in der Darstellung sind mit reinen HTML-Elementen nicht zu erreichen. Desweiteren bietet HTML – mit Ausnahme des Übertragens von Formulardaten – keinerlei Interaktionspotenzial. Ferner sind klassische HTML-Seiten, wenn sie einmal an den Browser übertragen wurden, bis zur nächsten Aktualisierung der Seite unveränderlich. Dynamische Prozesse, die ein regelmässiges *Update* einer Seite (oder von Teilen einer Seite) erfordern, sind in HTML bzw. in dem zugrundeliegenden Protokoll (HTTP) nicht vorgesehen. Um dem wachsenden Bedarf an solchen Möglichkeiten zu begegnen, haben sich in jüngster Vergangenheit neue Technologien etabliert, welche die genannten Einschränkungen aufheben und das Interaktionspotenzial von Webseiten und Web-Applikationen deutlich erhöhen. Als Schlüsseltechnologie ist hier AJAX zu nennen, das bereits an früherer Stelle in dieser Arbeit vorgestellt wurde (siehe Seite 20). Durch AJAX werden Webseiten und Web-Applikationen in ihrer Präsentation "reichhaltiger" (englisch: *richer*) – sie werden also zu *Rich Internet Application*. Empfehlungen zur zeitgemäßen Gestaltung von Benutzerschnittstellen im Web – wie sie z.B. in [SN09] beschrieben sind – lassen sich ohne die Hinzunahme neuer Technologien gar nicht realisieren.

Der Begriff *Rich Internet Application* (RIA) steht fortan im Mittelpunkt der Betrachtungen dieser Arbeit. Das vorliegende Kapitel führt ihn ausführlich ein und präsentiert den Status Quo an Technologien, die aktuell zur Entwicklung von RIAs zur Verfügung stehen. Eine besondere Beachtung erfährt dabei das Problemfeld der Barrierefreiheit im Zusammenhang mit diesen aktuellen RIA-Technologien. Basierend auf den Ausführungen dieses Kapitels präsentiert Kapitel 5 dann eine Erweiterung des jLAF-Frameworks, mit deren Hilfe Inhalte informationszentrierter RIAs in eine barrierefreie HTML-Version selbstständig überführt werden können. Offene Probeme bei vorrangig interaktionszentrierten RIAs sowie entsprechende Lösungsvorschläge sind dann Gegenstand des übernächsten Kapitels.

4.1.1 Zum Begriff der "Reichhaltigkeit"

Tim O'Reilly hat im Jahr 2005 in seinem Artikel "What is Web 2.0" (siehe [O'R05]) erstmals den Begriff des "Web 2.0" geprägt. Er nennt in diesem Artikel sieben Merkmale, an denen er die Veränderungen im Umgang mit dem World Wide Web im Vergleich zu dessen Ursprüngen festmacht. Das siebte Merkmal ist überschrieben mit *"Rich User Experiences"* und prognostiziert, dass in den nächsten Jahren[1] die klassischen Desktop-Applikationen zurückgehen und stattdessen als Web-Applikationen zur Verfügung stehen werden. Dies sei deshalb möglich, weil moderne Web-Techologien die Entwicklung von Internet-Anwendungen erlauben, die in Darstellung und Interaktionspotenzial ihren Desktop-Pendants in nichts mehr nachstehen und eben diese *Rich User Experience* anbieten können. Somit werden Web-Anwendungen zu *Rich Internet Applications* (RIAs).

Doch was genau macht RIAs attraktiv? Worin unterscheiden sie sich von bisherigen Web-Anwendungen? Was wird mit dem Attribut "reichhaltig" exakt adressiert? Exemplarisch für eine Vielzahl gleichlautender Antworten auf diese Fragen stehen die nachfolgenden drei Zitate:

- "Rich Internet Applications (RIAs) offer a rich, engaging experience that improves user satisfaction and increases productivity." (Zitat von der Adobe-Webseite)

[1] Man beachte, dass der Artikel bereits im Jahr 2005 erschienen ist.

- "Rich Internet Applications (RIAs) attract site owners attention because users like them[...]" [Rog08]
- "The majority of online consumers who have used RIAs say these applications improve their Web experiences – primarily because they find RIAs easy to use."[Rog07]

Das durch diese drei Zitate zum Ausdruck gebrachte Ziel bei der Konzeption und Entwicklung einer RIA besteht demnach darin, dass diese den Benutzer emotional anspricht, sich (in für Web-Anwendungen bislang ungewohnter Weise) produktiv benutzen lässt und dabei noch Spaß bei der Bedienung macht. Die reine Vermittlung von Inhalt ist nicht mehr die ausschließliche Motivation eines Web-Auftritts. Flankiert wird sie demnach durch den Wunsch, Besuchern (oftmals sind dies potenzielle Kunden) ein besonderes Erlebnis beim Besuch bzw. der Benutzung der Seite zu bieten.

4.1.2 Vorteile von RIAs

Zur Erreichung dieser Ziele warten RIAs – im Unterschied zu traditionellen Web-Auftritten – mit folgenden Vorteilen auf (wobei nicht jede RIA stets alle Vorteile zugleich erfüllt):

Bessere Kommunikation mit dem Server: Neben HTTP stehen in RIAs auch weitere, binäre Formate (z.B. das *Action Message Format*, AMF) zur Übertragung von Inhalten zur Verfügung. Mit ihnen kann insbesondere ein spürbarer Geschwindigkeitsvorteil bei der Übertragung großer Datenmengen erreicht werden.

Unabhängigkeit von Betriebssystemen und Browsern: RIAs werden nicht für ein spezielles Betriebssystem oder einen Browser(typ) entwickelt, sondern für eine Zielplattform. Im einfachsten Fall bedeutet dies die Kombination aus HTML, CSS und JavaScript. Weitreichendere Möglichkeiten stehen RIAs aber (erst) dann zur Verfügung, wenn sie tatsächlich eine eigene Ziel*plattform* adressieren. (Beispiele dafür werden in Kürze vorgestellt.) Steht die Zielplattform auf einem System zur Verfügung, so kann die RIA darauf unmittelbar ausgeführt werden. Dies reduziert den Entwicklungs- und Testaufwand, weil nicht mehr unterschiedliche Konfigurationen berücksichtigt werden müssen. Dies ist nicht nur aus Entwicklersicht von Vorteil, sondern auch für den Benutzer: auch er muss nicht (je Betriebssystem) verschiedene Versionen pflegen und kann sich auf eine einheitliche Darstellung der RIAs auf allen Systemen verlassen.

Verarbeitungsgeschwindigkeit durch eigene Logik: RIAs laufen auf dem Clientrechner und können dessen Rechenleistung verwenden. Somit ist es möglich, dass sie über eine eigene Logik verfügen. Deshalb muß nicht jede Anfrage an den Server gesendet, dort verarbeitet und dann wieder zurückgeschickt werden, sondern die Aufbereitung von Daten kann stattdessen direkt auf dem Clientrechner erfolgen. Dadurch lässt sich oftmals eine verbesserte Verarbeitungsgeschwindigkeit erreichen.

Reduzierung des Netzwerkverkehrs: Wenn Programmlogik lokal in der RIA liegt und nicht für jede Aktion ein Anfrage-Antwort-Zyklus durchlaufen werden muss, reduziert dies den notwendigen Netzwerkverkehr und führt letztendlich ebenfalls (abhängig von der zur Verfügung stehenden Bandbreite) zu einer Erhöhung der Arbeitsgeschwindigkeit.

Offline-Verwendbarkeit: Einige RIA-Technologien ermöglichen die Verwendung der Applikation auch dann, wenn keine Online-Verbindung besteht. Daten können lokal zwischengespeichert werden, bis der Rechner wieder online ist. Dies macht RIAs interessant für

den mobilen Einsatz, denn sie können offline (beispielsweise im Zug, im Flugzeug oder allgemein an Orten, an denen kein Internetzugang zur Verfügung steht) verwendet werden und synchronisieren die im Offline-Modus bearbeiteten Daten, sobald sie wieder eine Online-Verbindung feststellen.

Reichhaltigkeit in den Bedienkonzepten: Bedienkonzepte wie "Drag-and-Drop", "Panning"[2] und ähnliche aus Desktop-Anwendungen bekannte Mechanismen werden durch RIA-Technologien nun auch in Web-Applikationen möglich.

4.1.3 Nachteile von RIAs

Den Vorteilen stehen jedoch auch manche Nachteile gegenüber, die der Einsatz von RIAs mit sich bringt:

Downloadzeit: RIAs liegen in einer kompilierten Form vor, d.h. sie werden vor ihrer Ausführung vollständig vom Server geholt. Je nach Umfang der Anwendung kann dadurch eine Größe von mehreren Megabytes erreicht werden, was je nach verfügbarer Bandbreite beim Benutzer unerwünscht lange Downloadzeiten zur Folge haben kann. Ein die Größe von RIAs nicht unerheblich beeinflussender Faktor ist zudem auch die Tatsache, dass Schriften und multimediale Elemente (z.B. Bilder, Audios und Videos) in die RIA eingebettet werden können. Dem Vorteil, alle Inhalte nach dem Download unmittelbar zur Verfügung zu haben, steht dann jedoch die entsprechend anwachsende Dateigrösse als Nachteil gegenüber.

Verfügbarkeit: RIAs werden – mit Ausnahme derjenigen, die auf einem JavaScript-Framework basieren – für eine Technologieplattform entwickelt, nicht jedoch für spezielle Betriebssysteme. Sofern die Technologieplattform für einen bestimmten Rechner bzw. ein bestimmtes Betriebssystem verfügbar ist, läuft auch die RIA auf diesem Rechner. Umgekehrt gilt aber: die Lauffähigkeit der RIA ist abhängig davon, dass die verwendete Plattform auf dem gewünschten Rechner verfügbar ist und dies auch zukünftig bleibt.

Ressourcen: Weil RIAs Logik vom Server auf den Client verlagern, beanspruchen sie auf diesem auch eine entsprechende zusätzliche Last. Hier ist sicherlich bei der Entwicklung von RIAs Umsicht geboten, damit der erwünschte Vorteil der Plattformunabhängigkeit nicht dadurch zunichte gemacht wird, dass die Anforderungen an die Hardware so hoch sind, dass nicht alle Geräte (z.B. Mobiltelefone) die erforderlichen Ressourcen aufbringen.

Sicherheit: Den Gewinn an *Rich Features*, wie beispielsweise den Zugriff auf das lokale Dateisystem, erkauft man sich mit einer Einbuße an Sicherheit. Einer "fremden" Web-Anwendung Zugriff auf lokale Ressourcen zu gewähren (insbesondere auf solch sensible Ressourcen wie das lokale Dateisystem) erfordert sowohl vom Entwickler als insbesondere auch vom Anwender eine erhöhte Sensibilität in Bezug auf Sicherheit[3].

Barrierefreiheit/Indizierbarkeit durch Suchmaschinen: Im Unterschied zu traditionellen Web-Anwendungen sind RIAs in Bezug auf Barrierefreiheit sehr problematisch. Dies

[2] Unter *Panning* versteht man das Verschieben des dargestellten Ausschnitts, wie zum Beispiel einer digitalisierten Straßenkarte bei Google Maps (`http://maps.google.de`)

[3] Ein in diesem Zusammenhang zu beobachtender Trend ist das digitale Signieren von RIAs. Auf diese Weise kann der Anwender zuverlässig verifizieren, aus welcher Quelle die RIA wirklich kommt. Auch wenn dadurch bei weitem nicht jegliches Risiko ausgeschaltet werden kann, so geht von RIAs aus vertrauenswürdigen Quellen wohl eine deutlich geringere Gefahr aus.

wird in den Abschnitten 4.4 und 4.5 ausführlich diskutiert. Mit nahezu identischen Argumenten wirken sich RIAs auch nachteilig auf die Erreichbarkeit (und damit die Indizierbarkeit) durch Suchmaschinen aus, denn diese benötigen (genau wie Screenreader) semantisch klar strukturierte Inhalte, um die Seiten indizieren zu können.

Die unterschiedlichen technischen Möglichkeiten zur Erstellung und Verwendung von RIAs sollen im nächsten Abschnitt vorgestellt werden.

4.2 Aktuelle RIA-Technologien

Aktuelle RIA-Technologien lassen sich stets einer der beiden folgenden Kategorien zuordnen:

Kategorie 1: RIAs, die auf HTML, CSS und JavaScript basieren und somit unmittelbar in einem Browser laufen; eine Installation einer eigenen Zielplattform ist nicht nötig.

Kategorie 2: RIAs, die auf einer Laufzeitumgebung basieren, die in Form eines *Plug-in* auf dem Zielrechner verfügbar sein muss.

Sofern (im Falle von RIAs der Kategorie 2) auf dem Zielrechner die entsprechende Laufzeitumgebung existiert, gilt für *alle* RIAs (unabhängig von ihrer Kategorie), dass sie keine oder höchstens eine minimale Installation auf dem Endgerät erfordern und damit "überall" und "sofort" verfügbar sind. Vom lokalen Arbeitsplatz-PC über Notebooks bis hin zum Mobiltelefon[4] soll(t)en RIAs von jeder Stelle der Welt, von der aus eine Verbindung ins Internet hergestellt werden kann, benutzt werden können und somit die Idee von "8A-Software" erfüllen:

Applications Able to deliver Access to Anyone Authorized Anytime, Anyplace on Any Device.

Weil RIAs der Kategorie 2 für eine Laufzeitumgebung und nicht für einen bestimmten Browser oder ein spezielles Betriebssystem entwickelt werden, präsentieren sie sich auf jedem Zielsystem stets in derselben Form. Kompatibilitätsprobleme, wie sie zum Beispiel zwischen unterschiedlichen Browsern existieren, treten dann nicht mehr auf. Dies ist einerseits für den Endbenutzer von Vorteil, andererseits aber auch für den Entwickler der RIA. Denn er entwickelt lediglich für die entsprechende Zielplattform und muss die Software einmalig für diese Zielplattform testen. Ferner können aufwändige Fallunterscheidungen zur Überwindung von Betriebssystem- und/oder Browserspezifika entfallen, wodurch sich abermals eine deutliche Reduktion des Entwicklungsaufwandes ergibt.

Die technische Funktionsweise von RIAs der Kategorie 2 skizziert Abbildung 4.1. Während traditionelle Web-Applikationen den Zielrechner (bzw. konkreter: den Browser auf einem Zielrechner) lediglich als Präsentations- und Interaktionsmedium verwenden, die gesamte Anwendungslogik jedoch auf dem Server verbleibt, befindet sich eine RIA (insbesondere jene der zweiten Kategorie) bildlich ausgedrückt *zwischen* Server und Browser. Der Browser stellt die

[4] An dieser Stelle verläuft zum Zeitpunkt der Entstehung dieser Arbeit die Grenze zwischen Realität und Marketing-Wunschdenken. Denn derzeit bringt die Mehrheit der verfügbaren mobilen Endgeräte – insbesondere Mobiltelefone – noch nicht die technischen Voraussetzungen mit, die eine Verwendung von RIAs in einer dem PC vergleichbaren Qualität erlauben würde. Jedoch ist seitens verschiedener Hersteller für die nächsten 12 Monate hier ein entsprechender Durchbruch angekündigt.

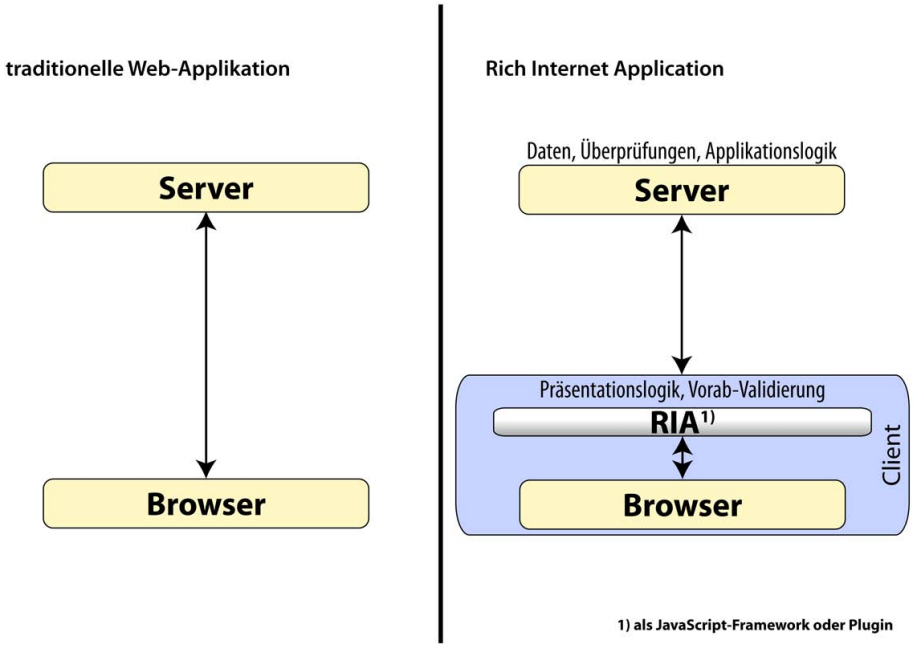


Abbildung 4.1: Gegenüberstellung von traditionellen Web-Applikationen und RIAs

Zielplattform bereit, bleibt ansonsten aber passiv. Für Präsentation, Interaktion und auch Logik bleibt allein die Zielplattform zuständig (im Falle der Logik jedoch insbesondere auch noch der Server). Ob der wesentliche Teil der Applikationslogik auf dem Server verbleibt oder in die RIA verlagert wird, ist eine Einzelfall-Entscheidung. Ein sinnvolles Anwendungsszenario ist in der Abbildung ausgedrückt: die Logik befindet sich auf dem Server, welcher von der RIA Daten (Benutzereingaben) empfängt und diese verarbeitet. Die RIA hingegen filtert durch ihre eigene Logik Falscheingaben heraus (Vorab-Validierung) und hilft auf diese Weise, unnötigen (weil fehlerhaften) Datenverkehr zwischen Client (RIA) und Server zu vermeiden. Ein gänzlicher Verzicht auf die serverseitige Validierung der gesendeten Daten darf aus Sicherheitsgründen nicht unterbleiben, weil immer die Gefahr besteht, dass ein Angreifer direkt (und nicht über die RIA) den Server kontaktieren und somit die clientseitige Validierung umgehen könnte (siehe hierzu auch die Ausführungen in Abschnitt 6.4 auf Seite 169).

Im folgenden sollen nun die aktuell am Markt verfügbaren Technologien zur Entwicklung und zum Betrieb von RIAs kurz vorgestellt werden.

4.2.1 JavaScript-Bibliotheken

Die Verwendung von HTML in Verbindung mit JavaScript (bzw. AJAX) und DOM-Scripting[5] erlaubt auf modernen Browsern die Darstellung beeindruckend interaktiver und insoweit "reichhaltiger" Seiten bzw. Web-Applikationen[6]. Entsprechend sind in der jüngeren Vergangenheit viele Bibliotheken erschienen, welche mit einer Vielzahl fertiger Interaktionselemente (*Widgets*) aufwarten und die Entwicklung solcher Seiten/Applikationen vereinfachen. Zum Teil werden

[5] DOM = Document Object Model; hierbei handelt es sich um eine Schnittstelle für den Zugriff auf HTML- und XML-Dokumente

[6] In diesem Zusammenhang hat sich auch der Begriff "dynamisches HTML" (DHTML) etabliert. Er beschreibt das Zusammenspiel von HTML, JavaScript und DOM-Manipulationen

diese Bibliotheken durch namhafte Unternehmen entweder direkt entwickelt oder zumindest finanziell unterstützt.

Begleitend zu ihrem Buch "*Designing Web Interfaces: Principles and Patterns for Rich Interactions*" unterhalten die Autoren Bill Scott und Theresa Neil eine Webseite, auf der sie 30 Widgets, die typisch für reichhaltige Anwendungen sind, daraufhin untersuchen, in welchen Frameworks sie enthalten sind und in welchen nicht[7]. Aus dieser Auflistung lässt sich aber implizit auch erkennen, welche der am Markt existierenden Frameworks von besonderer Bedeutung sind (vgl. dazu auch Anhang 8.2).

Die Grundidee ist hinter allen JavaScript-basierten Frameworks stets gleich: die Webseite wird durch JavaScript-Programme angereichert, welche für die Interaktionselemente auf der Seite verantwortlich sind. Durch DOM-Manipulationen und asynchrone Anfragen erlauben sie die Darstellung und Verwendung der Web-Applikation dergestalt, als handele es sich dabei um eine Desktop-Anwendung. Die Dreiteilung in Struktur (HTML), Präsentation (CSS) und Verhalten (JavaScript), wie sie bereits in Abbildung 2.1 auf Seite 10 zum Ausdruck gebracht wurde, lässt sich mit JavaScript-Frameworks sehr schön erreichen.

Für den Einsatz von JavaScript-Bibliotheken spricht, dass alle modernen Browser JavaScript unterstützen. Somit muss der Besucher einer solchen Seite nicht erst ein Plug-in installieren, um mit ihr interagieren zu können. Andererseits stellt die Vielzahl von verschiedenen Browser- und Betriebssystemkombinationen eine Herausforderung für den Einsatz solcher Bibliotheken dar. Solange nicht allen gängigen Browsern eine hundertprozentige Konformität für die Einhaltung von Web-Standards bescheinigt werden kann, sind im Einzelfall Unterscheidungen hinsichtlich des verwendeten Browsers im Programmtext notwendig. Dieses Problem wird umso größer, wenn man bedenkt, dass auch in sehr vielen Systemen noch ältere Browser benutzt werden, die für ihre mangelhafte Einhaltung von Standards bekannt sind. Hier besteht die Gefahr, dass interessierte Besucher der Seite, die mit einem solchen Browser arbeiten, einen völlig falschen Eindruck von der Seite erhalten oder im schlimmsten Fall gar nichts zu sehen bekommen. Zwar ließe sich das Problem durch ein Update des Browsers auf die neueste Version schnell lösen, jedoch liegt die Bereitschaft zur Installation eines neuen Browsers bei vielen Nutzern sicherlich bei weitem nicht so hoch wie diejenige zu einer Installation eines Browser-Plug-in, mit Hilfe dessen RIAs der Kategorie 2 verwendet werden können. Die folgenden Technologien fallen jeweils in diese (zweite) Kategorie.

4.2.2 Adobe Flex und Adobe AIR

Der kalifornische Software-Hersteller Adobe hat mit "Flex" das zum Zeitpunkt der Entstehung dieser Arbeit leistungsfähigste und bestunterstützte Framework zur Entwicklung von RIAs veröffentlicht. Adobe Flex[8], das derzeit in Version 3 vorliegt, ist betriebssystemübergreifend und quelloffen verfügbar. Zielplattform ist hier entweder der Flash-Player oder sein Stand-Alone-Äquivalent "Adobe AIR". Gleichwohl lassen sich mit Flex aber auch auf HTML- und AJAX basierende Lösungen entwickeln.

Für die Programmierung der Applikationslogik kommt in Flex die objektorientierte Programmiersprache ActionScript, welche dem ECMAScript-Standard entspricht und somit eine sehr große Ähnlichkeit zu JavaScript aufweist, zum Einsatz. Zur deklarativen Definition der Benutzerschnittstellen (GUI) steht die auf XML basierende Beschreibungssprache MXML zur Verfügung.

[7] Diese Seite ist zu finden unter: `http://designingwebinterfaces.com/essential_controls`

[8] `www.adobe.com/de/products/flex/`

Flex-Applikationen können entweder für den Flash-Player oder als sogenannte "AIR"-Applikation entwickelt werden. Versionen für den Flash-Player werden auf einem Webserver zur Verfügung gestellt (in der Regel als `swf`-Datei) und beim Aufruf im Browser-Plug-in ausgeführt. Eine Installation der Applikation ist nicht nötig. Mittlerweile existieren auch erste mobile Endgeräte, die das Flash-Format (und nicht nur die eingeschränkte Version "Flash light") unterstützen. AIR-Applikationen hingegen benötigen auf dem Zielrechner eine installierte AIR-Laufzeitumgebung[9], welche ebenfalls für gängige Betriebssysteme kostenlos verfügbar ist. AIR-Applikationen laufen jedoch nicht in einem Browser und erfordern eine minimale Installation auf dem Zielrechner. Dies macht sie im Unterschied zu Applikationen, die für den Flash-Player entwickelt wurden, etwas unflexibler. Als Ausgleich bieten AIR-Applikationen jedoch vielfältigere Möglichkeiten auf dem Zielrechner, insbesondere den Zugriff auf das dortige lokale Dateisystem. Außerdem bringt die AIR-Laufzeitumgebung eine leichtgewichtige relationale Datenbank (*SQLite*) mit und erlaubt so sehr umfangreiche Datenhaltung und -verarbeitung auf dem lokalen Rechner. Mehrere Medien, unter ihnen auch die New York Times[10], kündigten an, dass gegen Ende des Jahres 2009 oder im Laufe des Jahres 2010 erste mobile Endgeräte auf den Markt kommen würden, welche ebenfalls in der Lage sein sollen, AIR-Applikationen auszuführen. Eine Grenze zwischen traditionellen Desktop-Anwendungen und AIR-Applikationen lässt nicht grundsätzlich ziehen. Am besten ist der Charakter von AIR-Applikationen beschrieben, wenn sie als Webanwendungen, die sich nahtlos in den Desktop einfügen, bezeichnet werden.

Aufgrund der technischen Reife wurde Adobe Flex als Referenzplattform für die Prototypen und Praxisbeispiele, die im Rahmen dieser Arbeit entstanden sind, ausgewählt. Deshalb werden die zitierten Quellcode-Ausschnitte (sofern nicht anders angegeben) in ActionScript oder MXML geschrieben sein.

4.2.3 Microsoft Silverlight

Microsoft verfolgt mit seinem Silverlight-Projekt[11] einen nahezu identischen Ansatz wie Adobe mit Flex. Auch in diesem Fall besteht die Zielplattform aus einem speziellen Browser-Plug-in, das jedoch völlig inkompatibel zu Flash ist. Eine mit Adobe AIR vergleichbare Stand-Alone-Lösung ist derzeit nicht vorgesehen. Silverlight steht zum Zeitpunkt der Erstellung dieser Arbeit in Version 3.0 zur Verfügung.

Genau wie Flex trennt auch Silverlight die Programmlogik von der Präsentationsschicht. Letztere wird in Silverlight in XAML (*"Extensible Application Markup Language"*) beschrieben, einer deklarativen, XML-basierten Sprache, die insoweit das Pendant zu MXML in Flex ist. Ein interessanter Unterschied zu Flex ergibt sich aber bei Betrachtung der Sprache(n) zur Entwicklung der Programmlogik: Silverlight ist nicht auf eine Sprache festgelegt, sondern erlaubt die Entwicklung in allen Sprachen, die im hauseigenen Framework ".NET" aggregiert werden. Dieser Ansatz verspricht, dass zukünftig .NET-Entwickler in großer Zahl unmittelbar für die Erstellung von RIAs bereitstehen, weil sie in "ihrer" Sprache bleiben können und nicht – wie in Flex – zunächst eine neue Sprache erlernen müssen. Darüberhinaus soll durch die Verwendbarkeit aller .NET-Sprachen aber auch eine sehr gute und einfache Anbindung der RIAs an bestehende .NET-Backend-Systeme ermöglicht werden.

[9] AIR ist dabei einerseits ein Akronym für *Adobe Integrated Runtime*, bei Umkehrung der Leserichtung andererseits aber auch das Akronym RIA.

[10] siehe Online-Ausgabe der New York Times vom 10. November 2008: `http://bits.blogs.nytimes.com/2008/11/10/coming-next-mobile-application-stores-for-the-masses/`

[11] `http://www.microsoft.com/silverlight/`

Inwieweit diese Argumente die Verbreitung von Microsoft Silverlight begünstigen, kann zum Zeitpunkt der Erstellung dieser Arbeit noch nicht beurteilt werden. Derzeit befindet sich Silverlight in einer sehr frühen Phase nach der Markteinführung und hat noch keine nennenswerte Bedeutung erlangt.

4.2.4 JavaFX

Eine weitere Technologie im Wettbewerb am RIA-Markt hat Sun Microsystems mit JavaFX[12] eingeführt. Auch JavaFX trennt die Interface-Beschreibung (hier ebenfalls in Form einer deklarativen, jedoch nicht XML-basierten Sprache namens *JavaFX Script*) von der Programmlogik. Zur Implementierung der Programmlogik steht das gesamte Spektrum der Java-Bibliotheken bis hin zu J2EE zur Verfügung. Darin sieht Sun Microsystems auch eine der Stärken des eigenen Ansatzes. Denn einerseits lässt sich somit eine nahtlose Integration mit Informations-Infrastrukturen erreichen, d.h. man kann bestehende Backend-Systeme unmittelbar weiterverwenden, andererseits hilft die Verwendbarkeit bekannter Technologien in JavaFX aber auch, erfahrene Java-Entwickler ohne nennenswerten Weiterbildungsaufwand für JavaFX-Projekte einzusetzen.

Eine weitere Stärke wird in der geräteübergreifenden Verwendbarkeit von JavaFX-Applikationen gesehen. Denn neben der traditionellen virtuellen Maschine (Java-VM) als Zielplattform steht mit "JavaFX Mobile" auch die Möglichkeit bereit, RIAs auf mobilen Endgeräten lauffähig zu haben. Nach Auskunft von Sun Microsystems können bereits heute mit JavaFX entwickelte RIAs auf 90% aller Endgeräte (d.h. Computer und mobile Geräte) laufen, was mithin exakt der von Sun Microsystems verfolgten Idee hinter RIAs entspricht: *"Write once, run anywhere"*.

Ebenso wie Flex sieht JavaFX vor, Applikationen entweder in einer eigenen Laufzeitumgebung im Browser auszuführen oder als Stand-Alone-Applikation. Weil die erste Entwickler-Version von JavaFX erst im Dezember 2008 offiziell zur Verfügung gestellt wurde, ist die Marktdurchdringung dieser Technologie zum gegenwärtigen Zeitpunkt (November 2009) noch sehr untergeordnet, und ebenso wie bei Microsoft Silverlight wird auch bei JavaFX erst mittelfristig eine Tendenz bezüglich der Akzeptanz dieses Frameworks erkennbar sein.

4.2.5 Open Laszlo

Eine schon deutlich länger verfügbare und damit ausgereiftere Alternative zur Entwicklung von RIAs stellt OpenLaszlo[13] dar. Dieses ehemals kommerzielle Produkt liegt seit einiger Zeit als kostenfreies Open-Source-Framework vor. Hinsichtlich der Trennung von GUI- und Programm-Entwicklung präsentiert sich OpenLaszlo analog zu seinen Wettbewerbern. Ein sehr interessanter Unterschied liegt jedoch bei der unterstützten Zielplattform vor: an dieser Stelle abstrahiert OpenLaszlo zunächst von dieser und bietet dann zum Zeitpunkt der Auslieferung (*"Deployment"*) zwei Alternativen: entweder Flash oder DHTML. Der entscheidende Vorteil dieses Ansatzes liegt damit darin, dass dieselbe RIA sowohl als Flash-Applikation als auch als (D)HTML-Version ausgeliefert werden kann.

Das Marketing-Versprechen von OpenLaszlo, DHTML- und Flash-Anwendungen gleicher Qualität zu erhalten, war in einer im Rahmen dieser Arbeit betreuten Diplomarbeit einer Evaluation unterzogen worden. Diese Diplomarbeit hat Lučić mit [Luč08] angefertigt. Lučić gelangt

[12] `http://www.javafx.com/`
[13] `http://www.openlaszlo.org/`

zu dem Resultat, dass die versprochene Unabhängigkeit von der Zielplattform nur dann erreicht werden kann, "wenn bei der Entwicklung auf dem kleinsten gemeinsamen Nenner aller Plattformen entwickelt wird." Die von Laszlo Systems angekündigte Unterstützung weiterer Zielplattformen (darunter Microsoft Silverlight) sieht Lučić kritisch: "Es ist zu erwarten, dass Entwickler, deren unter OpenLaszlo entwickelte Anwendungen auf allen diesen Zielplattformen einsatzfähig sein sollen, nicht umhinkommen werden[,] sich mit bestimmten Eigenheiten der Zielplattformen auseinander[zu]setzen."

Vorteilhaft ist der hinter OpenLaszlo stehende Ansatz insoweit, als er eine wirkungsvolle Möglichkeit darstellt, dem generellen Problem der Barrierefreiheit in RIAs zu begegnen. Diesem Thema widmet sich der folgende Abschnitt.

4.3 Usability versus User Experience: Das Dilemma von RIAs

Zu Beginn dieses Kapitels wurde die Beschäftigung mit Rich Internet Applications unter anderem durch ein Zitat aus [Rog07] motiviert. Die Fortsetzung des Zitats im Originaltext macht auf einen neuen Aspekt bei der Betrachtung von RIAs aufmerksam:

> "[The majority of online consumers who have used RIAs say these applications improve their Web experiences primarily because they find RIAs easy to use.] To take advantage of this enthusiasm, firms need to provide clear paths to RIAs, make sure they are free of well-known usability problems, and deploy RIAs when they are likely to have the best impact."

Der Report bescheinigt RIAs demnach, wohlbekannte Probleme mit der *Usability* aufzuweisen. Zunächst erscheint eine Klärung des Begriffs *Usability* sowie des oftmals damit in Zusammenhang gebrachten Begriffs *User Experience* sinnvoll:

4.3.1 Usability

Der englischsprachige Begriff "Usability" lässt sich in die deutschen Sprache am besten mit "Gebrauchstauglichkeit" oder "Benutzerfreundlichkeit" übersetzen. Da dieser Anglizismus aber sogar im Duden aufgeführt ist, wird er im folgenden weiter verwendet. Usability bezieht sich noch nicht alleine auf Software oder Internetseiten, sondern allgemein auf alle Gegenstände und Einrichtungen des täglich Lebens, die Menschen zur Handhabung zur Verfügung stehen. Im umgangssprachlichen Verständnis steht die Usability als ein Indikator dafür, wie einfach und verständlich ein Produkt (hier: eine Web-Applikation oder eine Webseite) benutzt werden kann.

Eine konkretere Definition des Begriffs Usability findet sich in zwei ISO-Normen. So wird er in der englischen Originalfassung der ISO-Norm 9241-11, welche mit "Anforderungen an die Gebrauchstauglichkeit – Leitsätze" überschrieben ist, wie folgt definiert[14]:

> "[Usability is] the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."

[14] Diese Definition bezieht sich nicht nur auf Software, sondern allgemein auf ein Produkt.

Eine zweite Defintion – diesmal jedoch mit einem klaren Bezug zu Software – findet sich in der ISO-Norm 9126:

> "[Usability is] the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions."

Eine nicht mehr streng wissenschaftliche, aber dem Untertitel des Buches gerecht werdende, am "Menschenverstand" ("*Common Sense*") orientierte Definition gibt Krug in seinem Werk über *Web Usability*:

> "Usability really just means making sure that something works well: that a person of average ability and experience can use the thing for its intended purpose without getting hopelessly frustrated."[Kru05]

4.3.2 User Experience

Ein weiterer Begriff, der im Zusammenhang mit Mensch-Software-Interaktion oftmals genannt wird, lautet "*User Experience*" (abgekürzt mit "UX") und meint das "Nutzungserlebnis", das ein Benutzer bei der Verwendung eines Produktes (auch: Software) empfindet. Was empfindet der Benutzer emotional? Werden seine Erwartungen erfüllt werden oder nicht? (Unterforderung? Überforderung?) Ist die Benutzung angenehm? Macht sie Freude? Durch derartige Fragen ist unmittelbar erkennbar, dass der Begriff *User Experience* einen ausschließlich subjektiven Charakter aufweist. Dennoch kommt der Berücksichtigung der *User Experience* gerade in modernen Web-Applikationen und in RIAs eine große Bedeutung zu. Wenn sich der Besucher eines Webshops emotional angesprochen fühlt, wird er vermutlich länger in diesem Shop verweilen oder gerne zu einem späteren Zeitpunkt erneut dorthin zurückkehren und somit mehr oder öfter dort konsumieren. Aber nicht nur kommerzielle Interessen motivieren die Berücksichtigung der *User Experience* – ein e-Learning-System wird vermutlich mehr Erfolg bringen, wenn dessen Benutzung Spaß macht, als wenn es nur mit trocken dargebotener Wissensvermittlung aufwartet. Dieses Argument führt gewissermassen den Gedanken fort, der in Zusammenhang mit den Vorteilen der Flash-Technologie für kognitiv eingeschränkte Menschen erwähnt worden war (vgl. Abschnitt 2.7.1.1 auf Seite 24f.).

Mithin sollte der Thematik der *User Experience* bei der Entwicklung von IT-Systemen allgemein, insbesondere aber auch von Webseiten und RIAs bewusst Berücksichtigung finden. Jesse James Garrett, der Begründer des Begriffs "AJAX", fasst dies in seinem Buch über User Experience im Web sehr schön zusammen:

> "The user experience development process is all about ensuring that no aspect of the user's experience with your site happens without your conscious, explicit intent."[Gar02]

4.3.3 Das Dilemma

Die Erklärungen der Begriffe "Usability" und "User Experience" führen auf ein Dilemma, welches RIAs aufweisen: einerseits soll(t)en RIAs ihrer Bestimmung nach einen Gewinn im Hinblick auf *User Experience* von Web-Applikationen darstellen. Andererseits weisen sie aber

unter Umständen Probleme mit der *Usability* auf, wie dies beispielsweise in dem diesen Abschnitt einleitenden Zitat von [Rog07] zum Ausdruck kommt. Denn während RIAs für einen nicht-eingeschränkten Benutzer einen Gewinn an Benutzerfreundlichkeit darstellen können, bleiben sie Menschen mit Behinderungen zum Teil vollständig unzugänglich – meistens deshalb, weil die Zielplattformen der RIAs (der Kategorie 2) assistiven Technologien wie beispielsweise Screenreadern ihre Inhalte nicht vermitteln können. Aber auch bei RIAs, die in Kategorie 1 fallen, sind Probleme mit der Barrierefreiheit evident.

Beispiele hierzu und mögliche Lösungsansätze werden in den folgenden Abschnitten – geordnet nach Technologien – kurz vorgestellt.

4.4 Barrierefreiheit und JavaScript-Frameworks

4.4.1 Das grundsätzliche Problem

In den Möglichkeiten, die JavaScript zur Manipulation der Seitenelemente bereithält, liegt ein positives Potenzial für die Optimierung derartiger RIAs auf Barrierefreiheit. Beispielsweise können mehrere CSS-Vorlagen zur Verfügung gestellt werden, und der Benutzer wählt dynamisch aus, welche ihm und seinem individuellen Sehvermögen am angenehmsten erscheint. Mit Hilfe der bereits erwähnten DOM-Manipulation wird dann das ausgewählte Stylesheet zur Layout-Grundlage der aktuellen Seite gemacht. An dieser Stelle setzt bereits die Idee der "Individualisierung" bzw. "Personalisierung" an, die im weiteren Verlauf dieser Arbeit noch eine tragende Rolle spielen wird (vgl. Kapitel 6).

Wenn der Einsatz von HTML, CSS und JavaScript über die Präsentation statischer Seiten hinausgeht und mit seiner Hilfe stattdessen die Funktionsweise vollwertiger Anwendungen realisiert werden soll, führt das sehr schnell zu Webanwendungen, die nicht mehr barrierefrei sind. Neben den typischen – bereits in Kapitel 2 beschriebenen – Problemen, die es beim Einsatz von JavaScript zu beachten gilt, müssen beim Einsatz von auf JavaScript-Frameworks basierenden RIAs auch die folgenden Punkte beachtet werden:

1. Die Interaktionsmöglichkeiten, die einige der typischen GUI-Elemente in JavaScript-Frameworks bieten, lassen sich durch Screenreader nicht mehr darstellen. Ein Beispiel hierfür ist eine *TreeView*-Komponente, die eine Baumstruktur visualisiert und knotenweise auf- bzw. zugeklappt werden kann.

2. Mit Hilfe von AJAX wird der Inhalt in einem Teilbereich des Fensters aktualisiert (d.h. verändert). Ein Screenreader kann dies nicht vermitteln, wenn er zu dieser Zeit einen anderen Teil des Inhalts ausgibt.

3. Der Sinn und Zweck einiger GUI-Elemente sowie deren augenblicklicher Zustand lassen sich von Screenreadern nicht wiedergeben. Dies betrifft zum Beispiel einen Schieberegler, der dazu dient, eine Einstellung im Bereich von 0 bis 100 Prozent vorzunehmen. Screenreader sehen ihn lediglich als Grafik und können in der Regel weder seinen Sinn noch seine aktuelle Position (d.h. den "eingestellten" Wert) wiedergeben.

4. Nicht alle HTML-Elemente können den *Fokus* erhalten, d.h. mit Hilfe von Tastaturkommandos aktiviert werden. Macht eine RIA Gebrauch von derartigen Elementen, so ist sie nur per Maus, nicht aber per Tastatur verwendbar und somit sowohl für Sehbehinderte als auch für Menschen, die keine Maus bedienen können, unzugänglich. Ein Beispiel für

ein solches HTML-Element ist das Bildelement (`img`), wobei es sich zugleich auch um ein Element handelt, das zur ansprechenden optischen Gestaltung der RIAs weitreichende Verwendung findet.

5. Um eine JavaScript-basierte Manipulation des Seiteninhalts zu erreichen, kommen anstelle typischer HTML-Auszeichnungselemente (wie beispielsweise `h1`, `h2`, `p` usw.) benannte (d.h. mit einer ID versehene) Bereiche (`div`) zum Einsatz. Damit verwässert jedoch die semantische Struktur der Seite, was zu einem sehr ähnlichen Problem wie dem in Zusammenhang mit der WYSIWYG-Gestaltung in Content Management Systemen beschriebenen (vgl. Seite 35) führt.

6. Zur optischen Dialoggestaltung werden oftmals Hintergrundgrafiken verwendet. Wenn ein Benutzer in seinem Betriebssystem den Modus "besonders kontrastreiche Darstellung" gewählt hat, werden Hintergrundgrafiken nicht mehr dargestellt. Somit kann die Anwendung unter Umständen trotz Mauseinsatzes und Sehvermögens nicht verwendet werden, weil wesentliche Kontrollelemente nicht mehr angezeigt werden und infolgedessen auch nicht verwendet werden können.

Neben diesen spezifischen Problemen darf ein grundsätzliches Problem nicht übersehen werden: manche Benutzer haben JavaScript deaktiviert – aus welchen Gründen auch immer. Wege und Möglichkeiten, in diesem Fall dennoch eine sinnvolle Darstellung der RIA zu ermöglichen, werden im nächsten Kapitel in Zusammenhang mit den Programmierparadigmen *Graceful Degradation* und *Progressive Enhancement* sowie der Idee des damit eng verwandten *Unobstrusive JavaScript* erläutert (siehe dazu Abschnitt 5.4.1 auf Seite 129ff.).

Die Probleme, die in den ersten 5 Punkten angedeutet werden, adressiert die WAI ARIA-Initiative des W3C (vgl. Abschnitt 4.4.3). Aber auch mit "Bordmitteln" können Schwierigkeiten bei der Barrierefreiheit vermieden werden. Anhand eines Lösungsansatzes für das Problem Nummer 6 soll dies illustriert werden:

4.4.2 Beispiel: Konflikt Hintergrundgrafik / Kontrastreicher Modus

Das nachstehende Beispiel zeigt eine mögliche Strategie zur Unterstützung des kontrastreichen Modus in einem auf JavaScript basierenden RIA-Framework. Es wurde in Anlehnung an die Dokumentation eines derartigen Frameworks[15] erstellt.

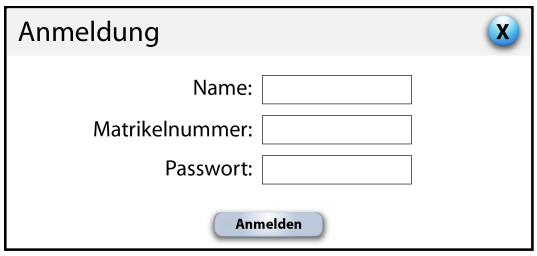


Abbildung 4.2: Typisches Dialog-Fenster in einer RIA

Abbildung 4.2 skizziert ein simples Dialogfenster. In dem Bestreben, RIAs ein Erscheinungsbild zu geben, das möglichst nah an demjenigen von traditionellen Desktop-Applikationen ist, enthält das Dialogfenster am oberen rechten Rand ein kleines Symbol. Klickt der Benutzer auf

[15] `http://www.dojotoolkit.org/book/dojo-book-0-9/part-2-dijit/a11y/dojo-accessibility-strategy`

dieses Symbol, so wird der Dialog ohne Verarbeitung der Eingaben beendet und das Fenster geschlossen. Realisiert wird ein solches Symbol oftmals durch Hintergrund-Grafiken. Diese werden im kontraststarken Modus des Betriebssystems jedoch nicht angezeigt. In der Folge würde einem solchen Dialog die Möglichkeit fehlen, ohne ein Versenden der Daten geschlossen zu werden. Abhilfe kann hier wie folgt geschaffen werden:

Schritt 1: Im HTML-Quellcode wird der entsprechende Teil so oder so ähnlich wie in Listing 4.1 gezeigt, lauten. In dem Bereich, in dem die Titelzeile beschrieben wird, muß innerhalb des `span`-Elements in Zeile 5 die folgende Ergänzung vorgenommen werden:

```
<span class="closeButton">X</span>
```

Schritt 2: In dem Stylesheet, das zur Darstellung der Seite verwendet wird, müssen die in Listing 4.2 genannten Zeilen ergänzt werden.

Schritt 3: Mit Hilfe eines JavaScript-Programms wird beim Laden der Seite überprüft, ob Hintergrundbilder dargestellt werden können oder nicht. Wenn ja, wird die Seite ohne weitere Manipulation zur Anzeige gebracht. Andernfalls ergänzt das JavaScript-Programm das `body`-Tag der Seite um das Attribut `class=".highContrast"`.

Listing 4.1: HTML-Code zur Erzeugung eines Dialogfensters (Titelzeile)

```
1 <div class="myDialog">
2    <!-- Beschreibung der Titelzeile -->
3    <div class="myDialogTitleBar">
4         <span class="myDialogTitle">Anmeldung</span>
5         <span class="myDialogCloseIcon" onclick="onCancel" />
6         </span>
7    </div>
8    <!-- ab hier: eigentlicher Inhalt des Dialogs -->
9 </div>
```

Listing 4.2: CSS-Code zur Erzeugung eines Dialogfensters (Titelzeile)

```
1 .myDialog .closeText {
2    display: none;
3    position: absolute;
4 }
5 .highContrast .myDialog .closeText {
6    display: inline;
7 }
```

Erläuterung: Die Zeilen 1 bis 4 in Listing 4.2 sorgen dafür, dass im normalen Modus die Ergänzung aus Schritt 1 nicht zur Anzeige kommt (`display: none`). Sofern die Klassen-Ergänzung aus Schritt 3 erfolgt ist, greift zusätzlich die Stylesheet-Definition aus den Zeilen 5 bis 7. Diese kaskadiert ("überschreibt") die Direktive aus Zeile 2 und sorgt somit dafür, dass der Text (hier: das "X") doch angezeigt wird (`display: inline`).

Das beschriebene Vorgehen bringt nicht nur Benutzern, die im kontraststarken Modus arbeiten, einen Vorteil, sondern auch jenen, die aus Sicherheitsgründen das Laden von Bildern unterbunden haben. Somit ist es darauf ausgerichtet, die Funktionalität einer Webseite bzw. Webapplikation in vollem Umfang auch dann aufrecht zu erhalten, wenn die Darstellung nur eingeschränkt möglich ist. Insoweit nimmt es schon mal die Idee des *Progressive Enhancement* vorweg, die im nächsten Kapitel eine nähere Vorstellung erfahren wird.

4.4.3 Die W3C-Initiative WAI-ARIA

Unter der Bezeichnung WAI-ARIA hat das W3C eine Initiative gestartet, welche eine Lösung der beschriebenen Probleme von JavaScript-basierten RIAs zum Ziel hat. WAI steht für *Web Accessibility Initiative*, und ARIA ist ein Akronym für *Accessible Rich Internet Applications*. Diese Initiative proklamiert als eines ihrer Ziele, neue Wege zu definieren, um die Funktionalitäten der RIAs auch assistiven Technologien bereitzustellen.

Im Kern besteht die WAI-ARIA-Initiative aus der Bereitstellung einiger Dokumente, unter denen sich Empfehlungen (*Best Practices*) sowie insbesondere eine Spezifikation für einen diesbezüglichen Standard befinden. Letztere liegt zum Zeitpunkt der Entstehung dieser Arbeit in einer Arbeitsversion (*Working Draft*) vom 24. Februar 2009 vor und ist mithin noch nicht endgültig verabschiedet.

4.4.3.1 Rollen, Zustände und Eigenschaften

Eine der wesentlichen Ideen hinter ARIA ist es, dass GUI-Elemente (in den Dokumenten als *Widgets* bezeichnet) mit zusätzlichen Attributen ausgestattet werden, um Auskunft über

- ihre Rolle in der Applikation (*Role*),
- ihren momentanen Zustand (*State*) sowie
- ihre momentanen Eigenschaften (*Properties*)

zu geben. Dies kann erneut anhand des bereits vorgestellten Beispiels eines Schiebereglers, der sich auf einen Wert zwischen 0 und 100 Prozent einstellen lässt, verdeutlicht werden:

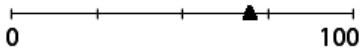


Abbildung 4.3: Eine typische Repräsentierung des GUI-Elements “Schieberegler”

Für einen sehenden Benutzer präsentiert sich ein solcher Schieberegler in einer ähnlichen Form wie in Abbildung 4.3 dargestellt. Letztlich ist er aber – im HTML-Sinne – nur eine Grafik, sodass ohne besondere Vorkehrungen ein Screenreader entweder nur das Wort “Grafik” oder – bei entsprechender Bezeichnung im Alternativtext-Attribut – beispielsweise “Grafik – Schieberegler” ausgeben würde. Ein Screenreader-Benutzer würde also weder über den Sinn des Reglers noch über dessen Eingabeintervall (hier: 0 bis 100) und auch nicht über dessen momentane Position informiert. Diese Informationen können im ARIA-Ansatz mit Hilfe von zusätzlichen Attributen dem HTML-Code hinzugefügt werden. Dies führt zum Beispiel zu einem HTML-Code wie in Codebeispiel 4.3.

Listing 4.3: Beispiel für ARIA-Ergänzungen bei einem Schieberegler

```
<input type="image"
       src="mySlider.gif"
       alt="Lautstaerke"
       role="slider"
       aria-valuemin="0"
       aria-valuemax="100"
       aria-valuenow="70"
       aria-valuetext="70 percent">
```

Erklärung zum Beispiel: Die ersten drei Zeilen beinhalten die traditionellen HTML-Angaben. Im Alternativtext-Attribut wird hier angegeben, was sich mit dem Schieberegler einstellen lässt, damit auch diese Information aus dem *Widget* selber hervorgeht und nicht nur durch den Fließtext in dessen Umgebung. Von Zeile 4 an beginnen die ARIA-Ergänzungen. Die Rolle, die der Grafik zuteil wird, ist diejenige eines Schiebereglers (*Slider*). Dieser kann die Werte von 0 bis 100 repräsentieren (Zeilen 5 und 6) und steht aktuell auf dem Wert 70 (Zeile 7). Dieser aktuelle Wert wird durch den Text in Zeile 8 (welcher den Wert übernimmt und um weitere Informationen angereichert werden kann, wie hier um das Wort "percent") vom Screenreader vorgetragen. Selbstverständlich müssen die ARIA-Attribute eine regelmäßige Aktualisierung erfahren, sobald sich der Zustand des Elements ändert. Mit Hilfe von JavaScript ist dies aber jederzeit problemlos möglich.

Auf eine Auflistung aller Rollen, Zustände und Eigenschaften, die für *Widgets* zur Verfügung stehen, soll an dieser Stelle verzichtet werden. Diese können in den entsprechenden Spezifikationen nachgelesen werden[16].

4.4.3.2 Live Regions

Schon mehrfach wurde in dieser Arbeit erwähnt, dass die partielle Aktualisierung von Seiten mit Hilfe von AJAX für Benutzer, die regulär die Web-Applikation verwenden, einen Zugewinn an Benutzerfreundlichkeit bringt. Gleichzeitig bewirkt sie das Gegenteil für Screenreader-Benutzer, denn Screenreader sind beim Vortragen auf einen Bereich der Seite *fokussiert*. Ändert sich Inhalt in einem anderen Bereich, so wird der Screenreader dessen nicht gewahr und der Benutzer demzufolge auch nicht. Alternativ müsste der Benutzer den aktuellen Bereich verlassen um zu prüfen, ob sich an anderer Stelle etwas verändert hat, und somit im Dokument hin- und herspringen.

Diesem Problem begegnet der ARIA-Ansatz durch die Idee der *Live Regions*. Bereiche innerhalb einer Seite, die häufigen Aktualisierungen unterworfen sind, können als eine *Live Region* gekennzeichnet und mit verschiedenen Parametern konfiguriert werden. Unter anderem erlauben diese Konfigurationsparameter zu beeinflussen, *aufgrund welcher Änderungen* diese annonciert werden (z.B. nur, wenn etwas hinzugefügt wird, oder nur, wenn etwas entfernt wird, oder immer, wenn sich Änderungen ergeben), in *welchem Umfang* einer Änderung annonciert wird (Der gesamte Bereich? Oder nur die Änderung selbst?) und in *welcher Art* dies geschieht. Letzteres meint, ob der aktuelle Lesefluss (im momentan fokussierten Bereich) sofort unterbrochen wird oder ob die Benachrichtigung erst erfolgt, wenn dies den Benutzer nicht unnötig ablenkt (z.B. am Ende des [Ab-]Satzes oder nach Beendigung einer Eingabe).

4.4.3.3 Navigation mit Hilfe der Tastatur

Sowohl Screenreader-Benutzer als auch Menschen mit motorischen Störungen sind auf die Verwendung der Tastatur (anstelle der Maus) angewiesen. HTML sieht aber nur bei wenigen Elementen vor, dass diese auch per Tastatur angesprochen werden (d.h. "den Fokus erhalten") können.

Die ARIA-Spezifikation hilft an dieser Stelle, indem sie *allen* sichtbaren Elementen diese Eigenschaft verschafft. Dazu kommt das Attribut `tabindex` zum Einsatz, das als Wert eine ganze Zahl erhält. Werte, die größer oder gleich 0 sind, können sukzessive mit Hilfe der Tastatur (in

[16] siehe Abschnitt 4.3ff. in `http://www.w3.org/TR/wai-aria/`

der Regel durch Drücken der Tabulator-Taste) angesprungen werden. Beginnend bei dem Element mit dem kleinsten nichtnegativen Tab-Index lassen sich so alle Elemente der gesamten Applikation durchlaufen. Wenn für Elemente ein Tab-Index kleiner als 0 vergeben wurde, dann sind diese Elemente nicht per Tastatur, sondern lediglich programmatisch (d.h. via JavaScript) erreichbar. Im Einzelfall kann es durchaus gewünscht sein, dass ein Element nicht in der Tab-Reihenfolge erscheint. Im Unterschied zu traditionellem HTML, das für viele Elemente keinen Tab-Index zulässt, kann dieser im ARIA-Ansatz also bewusst unterbunden werden.

4.4.3.4 Document Landmark Roles

Mit dem Konzept der *Document Landmark Roles* soll es assistiver Technologie ermöglicht werden, die Dokumentstruktur der Webseiten bzw. -applikationen besser zu erfassen. Es sieht vor, das HTML-Element `div` um ein Attribut mit der Bezeichnung `role` zu ergänzen. Dieses Attribut soll dazu dienen, die *Rolle* dieses Bereichs zu beschreiben, also eine Art Meta-Information über dessen Inhalt zu liefern. Die Werte, die diesem Attribut zugeordnet werden können, sind dabei jedoch nicht beliebig, sondern auf (derzeit) sieben begrenzt:

Article: Inhalt, der auch ohne den Rest der Seite – also für sich alleine genommen – bereits Sinn ergibt

Banner: auf den gesamten Webauftritt (d.h. auf die Website[17]) bezogener Inhalt, z.B. eine markante Titelzeile mit Logo

Complementary: Zusätzlicher Inhalt, der den eigentlichen Seiteninhalt ergänzt bzw. unterstützt, für sich alleine genommen aber ebenfalls Sinn ergeben würde. Dies kann zum Beispiel ein kleines Feld innerhalb eines Web-Portals sein, das den aktuellen Wetterbericht anzeigt.

Contentinfo: Informationen über den Inhalt selbst (z.B. Copyright-Informationen, Fußnoten, etc.)

Main: Der wesentliche Inhalt, der mit der Seite/Applikation transportiert werden soll

Navigation: ein Bereich, der Navigationselemente enthält

Search: ein Bereich, der ein Suchfeld enthält

Nicht jede Seite muss zwingend alle diese Bereiche aufweisen, und einige Rollen dürfen durchaus – solange sich dies nicht nachteilig auf das Erfassen der Dokumentstruktur auswirkt – auch mehrfach verwendet werden.

Screenreader können diese Rollen dann berücksichtigen und ausgeben, um somit dem Benutzer stets zu vermitteln, in welchem Bezug der aktuell fokussierte Inhalt zur gesamten Seite bzw. Applikation steht.

[17] vgl. zur Begriffsbestimmung die Fußnote auf Seite 31

4.4.3.5 Bedeutung für statische Webseiten

Die Ideen der WAI-ARIA-Initiative kommen nicht nur RIAs zugute, sondern können ebenfalls auf statische (informationszentrierte) Webseiten angewandt werden. Durch Angabe der *Document Landmark Roles* würde zum Beispiel der Aufbau der Seiten im WebTool (vgl. Abbildung 3.1 auf Seite 42) deutlich nachvollziehbarer für Screenreader. Die dort gezeigte Anordnung der einzelnen Seitenbereiche führt zu den folgenden Rollen:

- Navigation (*Navigation*),
- Titelzeile (*Banner*) und
- Inhaltsbereich (*main* oder *Article*).

Im WebTool bzw. jLAF (Kapitel 3) werden die einzelnen Seitenbereiche durch verschiedene `div`-Umgebungen repräsentiert. Einerseits kann die Unterstützung der *Document Landmark Roles* unmittelbar vom WebTool (bzw. durch das jLAF) in den HTML-Quelltext geschrieben werden, andererseits lassen sich die notwendigen Ergänzungen im HTML-Dokument aber auch mit Hilfe einer beim Laden der Seite ausgeführten JavaScript-Funktion nachträglich erreichen (siehe Listing 4.4). Am Quellcode bzw. dessen Generator müssen deshalb keine Änderungen vorgenommen werden.

Listing 4.4: Dynamisches Hinzufügen von *Document Landmark Roles* mit JavaScript nach [Lem08]

```
1  // Funktion zum Hinzufuegen des ARIA-Attributes 'role'
2  function addDocumentLandmarkRole(id, role)
3  {
4     var myElement = document.getElementById(id);
5     if (myElement)
6     {
7        myElement.setAttribute('role', role);
8     }
9  }
10
11 // die drei Bereiche hinzufuegen
12 function addLandmarks()
13 {
14     addDocumentLandmarkRole('content', 'main');
15     addDocumentLandmarkRole('nav', 'navigation');
16     addDocumentLandmarkRole('title', 'banner');
17 }
18
19 // beim Laden der Seite ausfuehren
20 window.onload = addLandmarks;
```

4.4.3.6 Nachteile des WAI-ARIA-Ansatzes

Grundsätzlich hat der WAI-ARIA-Ansatz keine negativen Auswirkungen auf die Seiten, in denen er zur Anwendung kommt. Allerdings führt er dazu, dass diese Seiten fortan nicht mehr als HTML 4.01- oder XHTML 1.0-konform validiert werden. Das W3C erarbeitet aber bereits Dokumenttyp-Definitionen, um an dieser Stelle Abhilfe schaffen zu können.

Zum Zeitpunkt der Erstellung dieser Arbeit ist der größte Nachteil noch der, dass die Spezifikation noch nicht endgültig verabschiedet ist und deshalb nur rudimentär Verbreitung und

Unterstützung findet. Die Hersteller bedeutender Browser und Screenreader haben ihre Absicht, die Spezifikation zukünftig zu unterstützen, jedoch bereits bekundet. Auch einige der in Anhang 8.2 genannten JavaScript-Frameworks bieten bereits Unterstützung von ARIA an.

4.5 Barrierefreiheit bei Plug-in-basierten RIAs

Die zuvor beschriebenen Probleme treffen grundsätzlich auch für RIAs der Kategorie 2 (Plug-in-basierte RIAs) zu, jedoch können die vorgestellten Lösungsstrategien nicht unmittelbar übernommen werden. Das Hauptproblem aller Plug-in-basierten RIAs ist, dass ihr Inhalt in einem eigenen Binärformat vorliegt und nicht in textueller Form (HTML). Ohne geeignete Vorkehrungen bleiben die Inhalte dieser RIAs für Screenreader in der Regel vollständig unsichtbar, d. h. im schlimmsten Fall würde ein Screenreader anstelle des Inhalts einer RIA lediglich ausgeben, dass die Seite ein "Medienelement" enthält.

Aus Sicht eines Browsers bzw. Screenreaders sind die Bestandteile der Benutzerschnittstellen der Kategorie-2-RIAs lediglich grafische Elemente. Selbst wenn ein solches Element noch näher identifiziert werden kann (z.B. als Kreis, Rechteck, etc.), so fehlt jegliche Semantik, die es einem Screenreader erlauben würde, die Bedeutung des Elements mitzuteilen. Ein Beispiel dafür skizziert Abbildung 4.4 (nach [Reg05]): die Abbildung demonstriert die viermalige Verwendung eines grafischen Elements (hier: ein simpler Kreis), welchem jedoch jeweils eine völlig unterschiedliche Bedeutung beigemessen wird. Die nachstehende Liste fasst die einzelnen Bedeutungen zusammen:

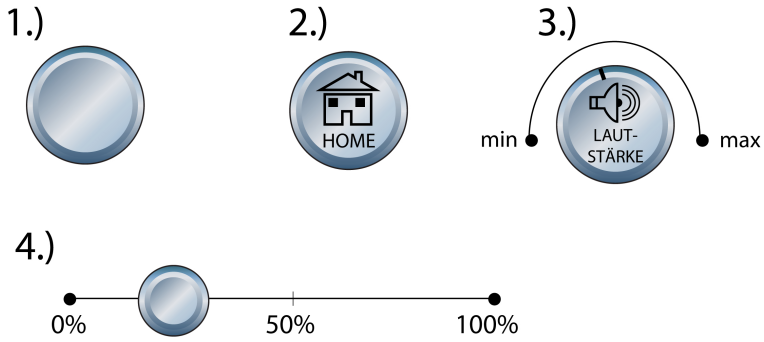


Abbildung 4.4: Unterschiedliche Bedeutungen eines simplen Kreises in Flash-Applikationen (nach [Reg05])

1.) ein simpler Kreis als grafisches Element – ohne weitere Funktion,

2.) ein Kreis, der als Schaltfläche (*Button*) dient (in diesem Beispiel als Link zur Homepage),

3.) ein Kreis, der als symbolischer Drehknopf dient, um die Lautstärke anzupassen,

4.) ein Kreis, der als "Anfasser" eines Schiebereglers (*Slider*) dient, mit dem sich hier eine Einstellung zwischen 0 und 100 % vornehmen lässt.

Einem sehenden Benutzer wird die jeweilige Bedeutung unmittelbar klar, ein Screenreader vermag sie jedoch nicht zu erkennen. Stattdessen müssen dem Element weitere Informationen zugeordnet werden, welche bei Bedarf dann durch die assistiven Technologien ausgewertet werden können.

Ein weiteres Problem kann der Einsatz von Farben in RIAs mit sich bringen. Die bereits weiter oben erwähnte Möglichkeit, eine besonders kontrastreiche Darstellung einzustellen, wirkt sich aktuell nicht auf die Browser-Plug-ins – und damit nicht auf RIAs der Kategorie 2 – aus. Diese RIAs müssen also idealerweise selbst eine solche Einstellmöglichkeit mitbringen. Der oftmals stark oder sogar rein grafische Charakter von RIAs mag auch Aufforderungen an die Benutzer beinhalten, welche eine Formulierung wie "Drücken Sie den grünen Knopf, um..." verwenden. Solche Aufforderungen sind für sehbehinderte Benutzer wertlos. Hilfreich werden sie frühestens dann, wenn sie zusätzlich noch eine Positionsangabe beinhalten (z.B.: "Drücken Sie den grünen Knopf oben links, um..."), aber selbst diese Angabe kann – wenn mehrere Buttons in unmittelbarer Nachbarschaft zueinander liegen – noch nicht präzise genug sein.

Die Hersteller der RIA-Frameworks und insbesondere auch die Entwickler, die mit ihnen arbeiten, haben die Problematik der Barrierefreiheit durchaus erkannt. Jedes der zuvor vorgestellten Frameworks adressiert das Problemfeld Barrierefreiheit mit unterschiedlichen Ansätzen und mit unterschiedlichen Intensitäten. Die Hersteller werben jedoch stets damit, barrierefreie RIAs zu ermöglichen. Die unterschiedlichen Ansätze der einzelnen Frameworks werden nachstehend kurz vorgestellt.

4.5.1 Barrierefreiheit mit Flex (sowie Flash und AIR)

Weil Flex-Applikationen den Flash-Player oder – im Falle von AIR – dessen Stand-Alone-Pendant verwenden, hängt die Frage, ob Flex die Entwicklung barrierefreier RIAs erlaubt, zunächst unmittelbar mit der Unterstützung der Barrierefreiheit im Flash-Player zusammen. Lange Zeit haftete Flash der Nimbus an, dass dessen Inhalte nicht barrierefrei zu verwenden seien. Zwar besaß die Flash-Technologie schon früh erste Ansätze, um den mit ihr erstellten *Rich Media* Inhalt auch für Screenreader zugänglich zu machen. In [Reg05] macht Regan aber bereits in der Einführung deutlich, dass diese Ansätze nur unter bestimmten Hard- und Softwarevoraussetzungen tatsächlich greifen. Hierbei darf jedoch nicht übersehen werden, dass zu der Zeit, zu der Regans Buch erschien (2005), Flash hauptsächlich zur Präsentation kleiner Animationen und Filme eingesetzt wurde, für die eine barrierefreie Alternativ-Repräsentation nicht von besonderer Bedeutung war. Das Angebot an Screenreadern war damals noch recht gering und machte den Einsatz des Betriebssystems Microsoft Windows notwendig. Insoweit begründete sich die Barrierefreiheit des Flash-Players in dessen Unterstützung von Microsofts "Active Accessibility"-Technologie (MSAA), einer API, die assistiven Technologien helfen soll, mit den Elementen der grafischen Benutzeroberfläche in vielfältiger Weise zu interagieren.

Mit der Weiterentwicklung von Flash ist auch dessen Unterstützung von Screenreadern deutlich verbessert worden. Auch Flex, das für die Entwicklung von RIAs eine wichtigere Bedeutung als das "reine" (animationszentrierte) Flash hat, sieht Möglichkeiten zur Repräsentation der Inhalte durch Screenreader vor. Flex-Applikationen sind aber ohne besonderes Vorgehen seitens der RIA-Entwickler nach wie vor per default *nicht* in der Lage, MSAA zu unterstützen. Vielmehr muß die entsprechende Unterstützung von MSAA zunächst durch Einstellungen im Compiler aktiviert werden. Die Begründung dafür, dass die Unterstützung nicht grundsätzlich aktiv ist, liegt in der Tatsache, dass die Dateigröße der übersetzten Dateien bei aktiver Unterstützung wächst. Dem allgemeinen Bestreben, die Dateigröße von RIAs so gering wie möglich zu halten, läuft dieser Umstand zuwider.

In der aktuellen Version von Flex bedeutet nach [Ado09a] eine aktive Unterstützung von MSAA, dass exakt 28 GUI-Elemente barrierefrei verwendet werden können, indem sie

- ein Textäquivalent bereitstellen,

- besonderen Einfluss auf die Positionierung von Beschriftungen und zugehörigen GUI-Elementen gestatten ("*Labeling Controls*") sowie

- eine Bedienung nicht nur per Maus, sondern auch per Tastatur ermöglichen.

Mit dem zweiten Punkt ist gemeint, dass erklärende Beschriftungen, die zu einem GUI-Element gehören, auch mit diesem zusammen als eine Einheit repräsentiert werden. Im zuvor erwähnten Beispiel eines Kreises, dessen Bedeutung durch Screenreader nicht zu vermitteln ist, wäre dies bereits eine Möglichkeit zur Lösung des beschriebenen Problems. Die *Labeling Controls* dienen aber auch dazu, die Reihenfolge zu definieren, in der die Elemente vom Screenreader vorgelesen werden. Anders als HTML-Seiten besitzen `swf`-Dateien keine typische Struktur im Sinne von "Von-links-nach-rechts" oder "Von-oben-nach-unten". Während sehende Benutzer den logischen Aufbau einer Benutzerschnittstelle erkennen können, kann ein Screenreader nur durch eine geschickte Reihenfolge bei der Wiedergabe des Bildschirminhalts für ein gutes Verständnis sorgen, bei ungeeigneter Reihenfolge leicht aber auch für Konfusion. Aus diesem Grunde lassen sich die 28 GUI-Elemente um das Attribut `tabOrder` ergänzen. Mit Hilfe dieses Attributs werden alle GUI-Elemente nummeriert und in aufsteigender Folge vom Screenreader wiedergegeben. Hier ist eine Parallele zur WAI-ARIA-Erweiterung aller Elemente um das Attribut `tabindex` erkennbar.

Aber auch Informationen über die RIA selbst, zum Beispiel eine Beschreibung ihrer eigentlichen Aufgabe sowie eine kurze Anleitung zu ihrer Bedienung, können von nichtsehenden Benutzern ohne besondere Vorkehrungen nicht erfasst werden. Eine Empfehlung von Adobe (vgl. [Ado09a]) sieht hierzu das folgende Vorgehen vor:

- Die Anwendung bekommt als erstes Element eines übergeben, welches eine Beschreibung über die Anwendung selbst enthält.

- Dieses Element wird so gestaltet, dass es für sehende Benutzer unsichtbar bleibt, weil es für diesen Benutzerkreis unbedeutend ist.

- Screenreader erfassen dieses Element jedes Mal, wenn der Benutzer die Anwendung startet oder an den Beginn der Anwendung zurückspringt. Dann wird der zugehörige Text ausgegeben.

- Das beschriebene Prozedere soll nur beim Einsatz eines Screenreaders zur Anwendung kommen. Deshalb werden die entsprechenden Erweiterungen der GUI-Elemente verwendet, die sich auf assistive Technologien beziehen. Für reguläre Browser sind diese Erweiterungen unbedeutend.

Die Code-Beispiele 4.5 und 4.6, die ebenfalls [Ado09a] entnommen wurden, demonstrieren dieses Vorgehen.

Listing 4.5: Beispiel für die Ergänzung einer Applikationsbeschreibung in Flex

```
1 function accessibleInit() {
2     var desc = createObject("TextInput","desc",0);
3     desc.width = 0;
4     desc.height = 0;
5     desc.x = 0;
6     desc.y = 0;
7     desc.accessibilityProperties = new AccessibilityProperties();
```

```
8     desc.accessibilityProperties.description= 'Access instructions. The Flex Blog
          Reader reads entries from different Blogs on the Web. To interact with
        this application, turn forms mode on after entering the application.';
9  }
```

Listing 4.6: Initialisierung der Applikation in MXML

```
1  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
2     mx:Application xmlns:mx=""
3     creationComplete="accessibleInit()"
4     pageTitle="My Accessible Flex Application">
```

Erläuterungen zum Codebeispiel: Codebeispiel 4.6 zeigt, wie unmittelbar nach der Erstellung eines Applikationsobjektes eine Funktion namens `accessibleInit()` aufgerufen wird. Diese Funktion wird in Listing 4.5 definiert und bewirkt das Folgende:

- In Zeile 2 wird zunächst eines der 28 Interface-Elemente, die eine Unterstützung der Barrierefreiheit mitbringen (ein Feld zur Eingabe von Text), instanziiert.
- Die Größe dieses Eingabefeldes wird in den Zeilen 3 und 4 auf den Wert 0 gesetzt. Dies bewirkt, dass dieses Feld zwar existiert, aber bei "normaler" Verwendung der Applikation auf einem Monitor nicht zu sehen ist.
- In den Zeilen 5 und 6 wird das neue Textfeld auf die linke obere Ecke des Anwendungsfensters positioniert. Diese Koordinaten gelten als Startposition innerhalb der Anwendung.
- In den Zeilen 7 und 8 werden nun die eigens auf Barrierefreiheit ausgerichteten Eigenschaften (`AccessibilityProperties`) dem Textfeld hinzugefügt und eine dieser Eigenschaften (die Beschriftung, engl. *description*) initialisiert.

Wenngleich das gezeigte Vorgehen aufgrund der gesetzten Breiten- und Höheneigenschaft keinerlei Auswirkungen auf diejenigen Benutzer hat, die regulär mit einem Monitor arbeiten, werden Screenreader das in der Funktion aus Listing 4.5 erstellte Textfeld-Objekt aufgrund dessen Positionierung als erstes Objekt der Anwendung erfassen. Da das Objekt ferner die Eigenschaften zur Barrierefreiheit unterstützt, werden diese (hier: die Beschreibungs-Eigenschaft) vom Screenreader vorgetragen. Im Resultat bedeutet das, dass mit jedem Start der Anwendung ein Benutzer eines Screenreaders zunächst darüber informiert wird, wozu die Applikation dient und wie sie verwendet werden kann.

Weitere Vorkehrungen, die in Flex getroffen werden müssen, um für Sehbehinderte eine weitgehend barrierefreie RIA zu erstellen, sind beispielweise die Verwendung von Textäquivalenten zu grafischen Elementen wie Bildern oder Videos. Dies entspricht im Wesentlichen den in Kapitel 2 beschriebenen Alternativtexten bei Bildern in HTML. Die vielfältigeren Möglichkeiten, die Flex im Unterschied zu HTML bietet, erlauben jedoch noch eine Unterscheidung: Textäquivalente können – ähnlich wie im Codebeispiel zuvor vorgeführt – auf den *Accessibility Properties* basieren. Dies macht sie *nur* für Screenreader auffindbar. Eine Verwendung der Eigenschaft `toolTip` würde hingegen dazu führen, dass jeder Benutzer dieser Textäquivalente gewahr werden kann, denn abgesehen davon, dass sie von einem Screenreader vorgetragen werden, kommen sie auch regulär zur Anzeige, wenn der Benutzer eine Zeit lang mit der Maus auf dem zugehörigen Element verweilt.

Die Berücksichtigung von Sehbehinderten und eine Ermöglichung der Verwendung von Screenreadern ist aber nicht die einzige Maßnahme, die es zu ergreifen gilt, um RIAs barrierefrei

zu gestalten. So muss beispielsweise auch sichergestellt sein, dass Menschen mit motorischen Behinderungen einen uneingeschränkten Zugang zu allen Funktionalitäten der Anwendung haben. Insbesondere muss die Anwendung auch dann zu verwenden sein, wenn keine Maus benutzt wird. Techniken wie *Drag 'n' Drop* sind somit also zunächst einmal *nicht* barrierefrei. Abhilfe kann hier durch den Einsatz redundanter *Events*[18] geschaffen werden: diejenige Logik, die beim Auslösen eines *Drag 'n' Drop*-Ereignisses ausgeführt wird, wird ebenfalls einem bestimmten Tastendruck-Ereignis zugeordnet. Somit kann die gewünschte Funktionalität wahlweise mit der Maus oder mit der Tastatur erreicht werden. Voraussetzung dafür, dass dieser Ansatz funktioniert, ist (bei Flex) jedoch, dass das GUI-Element in der Lage ist, mit Tastatur-Ereignissen umzugehen. Im Falle des Elements `image`, das zur Darstellung von Bildern dient, trifft dies zum Beispiel – analog zum `img`-Element in HTML (vgl. Seite 91) – *nicht* zu. Wenn die Anwendung also die Möglichkeit bereitstellen soll, Bild-Elemente via *Drag 'n' Drop* zu verschieben, lässt sich eine barrierefreie Tastatur-Alternative dazu deshalb nicht realisieren, weil Bild-Elemente keine Tastatur-Ereignisse unterstützen. Werden anstelle der Bild-Elemente hingegen Links verwendet, kann das Gewünschte erreicht werden: Objekte der Klasse `LinkButton` können einerseits sowohl mit der Maus als auch mit der Tastatur angesprochen werden, sind zudem aber andererseits auch in der Lage, sich nicht nur textuell, sondern in grafischer Form zu präsentieren. Mithin gibt es keinen visuellen Unterschied in der Anwendung, wenn anstelle von `Image`-Objekten `Link`-Objekte verwendet werden. Hingegen kann die Web-Seite bzw. -Anwendung durch dieses Vorgehen im Hinblick auf Barrierefreiheit profitieren, während sie bei Vorgehen Nummer 1 (Verwendung von `Image`-Objekten) automatisch als nicht mehr barrierefrei gewertet werden muss.

Diese Beispiele – die sich speziell auf die Flash-Technologie beziehen – machen deutlich, dass die Aussage, auf Flash basierende RIAs seien barrierefrei, per se nicht zutreffend sein muss. Aber auch das Gegenteil ist deshalb nicht automatisch der Fall. So unterstützen die neueste Version von Flash sowie Flex und AIR zwar Barrierefreiheit und assistive Technologien, aber nur eine sorgfältige, sich des Problemfeldes bewusste Programmierung wird auch dazu führen, dass die damit erstellten Medien wirklich barrierefrei zu verwenden sind. Die Verantwortung dafür trägt demnach der Entwickler. Und so mahnt auch der Hersteller von Flex (Adobe) auf der entsprechenden Webseite:

> *"For the Flex designer and developer, the challenge of accessible design is to remove any obstacles that prevent assistive technologies such as screen readers or keyboard-only tools from functioning effectively when rendering Flex RIAs. Achieving this goal ultimately requires designers and developers to pay special attention to the potential use cases for disability and design with these particular experiences in mind."*[Ado09b]

Aktuell unterstützen Flex-Applikationen nur den kommerziellen Screenreader JAWS, der zudem auch nur für Windows verfügbar ist. Überdies müssen noch einige JAWS-Skripte installiert werden, bevor die Interoperabilität von Flex(-Applikationen) und dem JAWS-Screenreader funktioniert[19]. Somit ist die barrierefreie Verwendung von Flex-Applikationen nur unter bestimmten Voraussetzungen und mit einem zusätzlichen Einrichtungsaufwand möglich. Zwar

[18] Mit dem Begriff *Event* werden in manchen Programmiersprachen – so auch in ActionScript – Ereignisse benannt, die zur Laufzeit auftreten. Zu den Ereignissen zählen insbesondere auch das Klicken mit der Maus sowie das Drücken von Tasten. Jedem Ereignis kann eine Funktion bzw. Methode zugeordnet werden, die immer dann zur Ausführung kommt, wenn das Ereignis eintritt. Derartige Funktionen werden als "*Event Handler*" bezeichnet.

[19] siehe `http://www.adobe.com/accessibility/products/flex/jaws.html`

arbeiten verschiedene OpenSource-Projekte auch an eigenen Ansätzen zur Barrierefreiheit von Flash-Inhalten, jedoch adressieren diese weniger Flash-/Flex-*Applikationen*, sondern eher multimediale Inhalte (Animationen).

4.5.2 Barrierefreiheit in Microsoft Silverlight

Auch in Silverlight findet das Thema Barrierefreiheit Berücksichtigung. Die Maßnahmen, die ein Entwickler ergreifen muss, um barrierefreie Silverlight-Applikationen zu erstellen, sind im Wesentlichen identisch zu denen, die zuvor in Zusammenhang mit Adobe Flex beschrieben wurden. Weil Silverlight von Microsoft entwickelt wird, weist es eine enge Verzahnung mit dem Betriebssystem Windows auf. Deshalb können Silverlight-Applikationen beispielsweise auch auf die Verwendung des kontraststarken Modus' geeignet reagieren. Hierzu setzt Silverlight auf die Programmierschnittstelle "User Interface Automation" (UIA-API), die von Microsoft beschrieben wird als "*new accessibility framework for Microsoft Windows, available on all operating systems that support Windows Presentation Foundation (WPF)*".

Nativ ist die UIA-API in Windows Vista und Windows 7 enthalten, für ältere Versionen des Betriebssystems stehen sogenannte *Bridges* zur Verfügung, die als Adapter zwischen der UIA-API und deren Vorgänger-API (der bereits erwähnte MSAA-API) verwendet werden können. Auch für Linux ist eine solche *Bridge* angekündigt[20].

In dieser Beschreibung liegt aber auch zugleich das größte Problem dieser Technologie – und damit auch von Microsoft Silverlight: UI Automation ist nur dort verfügbar, wo es vom Betriebssystem unterstützt wird. Auf aktuelle Versionen von Microsoft Windows trifft dies zwar zu, aber für Silverlight hat dies zur Konsequenz, dass diese RIAs nur dann barrierefrei verwendet werden können, wenn sie auf einem Rechner mit dem entsprechenden Betriebssystem bzw. unter Verwendung des erwähnten Adapters ausgeführt werden.

4.5.3 Barrierefreiheit in JavaFX

In Form der *Java Accessibility API* steht eine ausgereifte Programmierschnittstelle zur Verfügung, um Java-Applikationen auch auf alternativen Ausgabegeräten verwenden zu können. Leider unterstützt JavaFX zum Zeitpunkt der Entstehung dieser Arbeit diese API – und damit insgesamt das Thema Barrierefreiheit – nicht nativ. Um eine zumindest rudimentäre Zugänglichkeit von JavaFX-RIAs zu ermöglichen, steht seit Oktober 2009 das Paket `fxaccessibility` der Entwickler Thomas Butter und Jens Arndt zur Verfügung[21]. Der Ansatz von Butter und Arndt nutzt dazu die Java-GUI-Bibliothek *Swing*, denn diese bietet eine Anbindung an die Java Accessibility API und ermöglicht so eine Unterstützung von Screenreadern.

Weil das Projekt von Butter und Arndt erst unmittelbar vor Fertigstellung dieser Arbeit vorgestellt wurde, kann noch keine Aussage über dessen Praxistauglichkeit getroffen werden. Von Sun Microsystems liegen aktuell keine Informationen zu einer geplanten nativen Unterstützung von Barrierefreiheit in JavaFX vor. Jedoch ist Sun Microsystems in das im Herbst 2008 begonnene EU-Projekt *Aegis*[22] involviert, das sich (unter anderem) die Verbesserung der Barrierefreiheit von RIAs zur Aufgabe gemacht hat und dabei soweit wie möglich auf OpenSource-Lösungen setzt. Aus diesem Grund steht zu vermuten, dass JavaFX mittelfristig Fortschritte bei der Unterstützung von Barrierefreiheit verzeichnen wird.

[20] `http://www.w3.org/WAI/GL/WCAG20/implementation-report/Silverlight_accessibility_support_statement`

[21] Projekthomepage: `http://code.google.com/p/fxaccessible/`

[22] Projekthomepage: `http://www.aegis-project.eu/`

4.5.4 Barrierefreiheit in OpenLaszlo

RIAs, die mit OpenLaszlo entwickelt werden, bleiben zunächst unabhängig von der späteren Zielplattform. Erst bei der Auslieferung (*Deployment*) kann entschieden werden, ob die mit OpenLaszlo entwickelte Anwendung im Flash-Player laufen soll oder als DHTML-Version in einem normalen (JavaScript-fähigen) Browser. Insoweit ergibt sich die Barrierefreiheit von auf OpenLaszlo-basierenden RIAs letztendlich aus der Frage, inwieweit die Zielplattformen Barrierefreiheit unterstützen.

Die Erläuterungen der vorstehenden Abschnitte machen deutlich, dass sowohl (D)HTML als auch Flash-Applikationen grundsätzlich barrierefrei verwendet werden können, und daraus folgt, dass dies auch für OpenLaszlo-RIAs gilt. Zur Entwicklung mit OpenLaszlo steht eine eigene API zur Verfügung, die – ebenfalls von der Zielplattform abstrahierend – Attribute und Methoden bereitstellt, um die in einem der vorangegangenen Abschnitte beschriebenen GUI-Elemente um diejenigen Attribute, die für Screenreader von Bedeutung sind, ergänzen zu können.

4.5.5 WAI-ARIA: User Agent Implementation Guide

Zur Unterstützung assisitiver Ausgabemedien greifen Flex und Silverlight auf Mechanismen zurück, die vom Betriebssystem bereitgestellt werden, jedoch variieren die diesbezüglichen Ansätze von Betriebssystem zu Betriebssystem. Im Fall von Windows und Linux gibt es zwar ein von Novell ins Leben gerufenes Projekt, um die *Accessibility APIs* dieser zwei Betriebssysteme interoperabel zu machen. Die bereits in Abschnitt 4.4.3 vorgestellte WAI-ARIA-Initiative des W3C geht aber noch einen Schritt weiter und sieht auch hierzu eine Standardisierung vor, um zukünftig die Abhängigkeit von bestimmten Plattformen überwinden zu können. In ihrem Dokument "WAI-ARIA 1.0 User Agent Implementation Guide"[23] schlägt die Initiative vor, wie die ARIA-Spezifikationen mit Hilfe geeigneter Programmierschnittstellen des Betriebssystems (*Accessibility API*) aus einer RIA heraus an assisitive Technologien übermittelt werden können. Zum Zeitpunkt der Entstehung dieser Arbeit liegt das Dokument aber erst in einer sehr frühen Entwurfsversion vor und macht bereits in Punkt 1.1 darauf aufmerksam, dass es noch keinerlei Informationen über aktuelle *Accessibility APIs* berücksichtigt.

4.5.6 Indizierbarkeit durch Suchmaschinen

Wenn plug-in-basierte RIAs auf Webservern bereitgestellt werden, liegen sie – wie beschrieben – in compilierter Form vor und haben somit keine nach außen hin sichtbare textuelle (semantische) Struktur mehr. Daraus folgt einerseits die erläuterte Problematik im Zusammenspiel von Screenreadern mit derartigen RIAs. Gleichzeitig ergibt sich aber auch das Problem, dass diese RIAs nicht mehr von Suchmaschinen indiziert werden können, weil auch diese keinen "Zugang" mehr zu den RIA-Inhalten haben. Zwar lässt sich die RIA als solches noch durch Suchmaschinen indizieren, jedoch ist es nicht oder nur sehr eingeschränkt möglich, die dargebotenen Inhalte zu analysieren.

Eine allgemeingültige Lösung für dieses Problem ist zum Zeitpunkt dieser Arbeit nicht vorhanden. Auch der Entwurf von HTML 5 (aktuell als *Working Draft*-Version in [W3C09] publiziert), der unter anderem eine bessere Medienhandhabung vorsieht, bietet hier keine Lösungsansätze.

[23] siehe `http://www.w3.org/TR/wai-aria-implementation/`

Tabelle 4.1: Vor- und Nachteile beim Einsatz von RIAs

Vorteile	Nachteile
Interaktionspotenzial und Gestaltungsfreiraum	Downloadzeit
Plattformunabhängigkeit (auch in der Entwicklung)	Verfügbarkeit der Laufzeitumgebung
kann eigene Logik enthalten	benötigt Ressourcen auf dem Client-Rechner
Reduktion des Netzwerkverkehrs	Sicherheit
"*Ubiquitous Computing*": keine Installation erforderlich, Einsatz auch auf mobilen Geräten möglich	Barrierefreiheit/Indizierbarkeit durch Suchmaschinen

Somit obliegt es den Herstellern der Plug-ins, geeignete Maßnahmen zur Verbesserung der Erreichbarkeit durch Suchmaschinen zu ergreifen. Schonfeldt beschreibt in [Sch08] den Ansatz, den das Unternehmen Adobe in Kooperation mit den Suchmaschinenbetreibern *Google* und *Yahoo!* unternimmt, um Flash-Inhalte indizierbar zu machen. Im Wesentlichen wird dazu ein spezieller Flash-Player eingesetzt, der selbstständig den gesamten Flash-Inhalt "besucht" und ihn dabei in eine für Suchmaschinen geeignete Form transformiert.

4.6 Zusammenfassung zu Rich Internet Applications

Das Internet wird sich zukünftig stärker aus Rich Internet Applications zusammensetzen als aus traditionellen Webseiten. Die Kombination aus ansprechender, intuitiver Bedienung und der Aktualität und Interaktivität von Webanwendungen macht den besonderen Reiz von RIAs gegenüber Desktop-Anwendungen und traditionellen Webseiten aus. Tabelle 4.1 stellt die in diesem Kapitel behandelten Vor- und Nachteile nochmals übersichtlich zusammen.

Die Art und Weise, in der die einzelnen RIA-Frameworks der Problematik hinsichtlich ihrer Barrierefreiheit begegnen, wird mit über ihren zukünftigen Markterfolg entscheiden. Die rege Diskussion in einschlägigen Benutzerforen über dieses Thema zeugt von einer existierenden Nachfrage nach barrierefreien Einsatzmöglichkeiten. Das WAI-ARIA-Framework erlaubt die Hinzunahme von Attributen zu Elementen einer auf JavaScript basierenden RIA, um somit den oftmals sonst nicht nachvollziehbaren Aufbau der Seite für assistive Technologien zu beschreiben. RIAs, die für eigene Laufzeitumgebungen entwickelt wurden, profitieren vom WAI-ARIA-Framework hingegen nicht. Die Bestrebungen der WAI, eine Standardisierung der Interaktion von RIAs mit der *Accessibility API* des Betriebssystems vorzunehmen, befinden sich momentan in einem noch sehr frühen Stadium und sind insbesondere noch nicht plattformunabhängig. Bis assistive Technologien nativ von den jeweiligen Zielplattformen unterstützt werden, empfiehlt sich deshalb, eine zugängliche Alternativ-Version der RIA in reinem HTML vorzuhalten. Fragestellungen und Lösungsansätze dazu sind Gegenstand des nächsten Kapitels.

Kapitel 5

HTML-Fallback-Lösungen für statische RIAs

"Web pages can be more than text,
but they should not be less *than text"[Kor02]*

5.1 Motivation und Begriffsbestimmungen

Die Ausführungen im vorausgegangenen Kapitel lassen vermuten, dass Webauftritte in Zukunft noch stärker als momentan von multimedialen Elementen Gebrauch machen und daher in Form von Rich Internet Applications angeboten werden. An die Stelle traditioneller Gestaltung mittels HTML und CSS treten dann interaktive(re) Designs auf Basis von RIA-Oberflächen.

Wenn RIA-Technologien nur deshalb Verwendung finden, weil sie ein anspechenderes, interaktiveres und multimedialeres Layout eines Webauftritts ermöglichen, dann handelt es sich bei derartigen Webauftritten nicht unmittelbar um Anwendungen (wie dies der Begriff "Rich Internet *Application* suggeriert). Der Gedanke, eine *Anwendung* anzubieten, ist dann gar nicht ausschlaggebend für den Einsatz der RIA-Technologie. Vielmehr sind es die übrigen Vorteile von RIAs, wie inbesondere die Gestaltungsfreiheit und Interaktivität, welche ausschlaggebend für den Einsatz dieser Techniken sind (siehe hierzu auch Abschnitt 4.1.2 auf Seite 81).

In Kapitel 4 wurde gezeigt, dass eine wirklich "rundum" barrierefreie (also nicht nur auf einzelne Beeinträchtigungen fokussierte) Gestaltung von RIAs aktuell noch nicht möglich ist. Gleichzeitig wurde mehrfach auf die – zum Teil auch gesetzlich geforderte – Notwendigkeit hingewiesen, barrierefreie und standardkonforme Webauftritte zu unterhalten. Betreiber von Webseiten, die sich um einen barrierefreien Auftritt bemühen, der für eine möglichst große und stark heterogene Menge von Benutzer zugänglich sein soll, haben deshalb – wenn sie RIAs einsetzen möchten – keine andere Wahl, als sowohl eine RIA- als auch eine (barrierefreie) HTML-Version vorzuhalten. Aus redaktioneller ebenso wie aus technischer Sicht ist es jedoch nicht empfehlenswert, zwei getrennte Versionen (eine RIA-Version *und* eine HTML-Version) eines Webauftritts manuell zu erstellen und somit auch pflegen zu müssen. Daraus ergibt sich die Problemstellung, ob und wenn ja *wie* eine barrierefreie HTML-Darstellung aus RIA-Inhalten gewonnen und idealerweise automatisiert generiert werden kann.

Genau mit dieser Problemstellung beschäftigt sich das vorliegende Kapitel. Mit dem in Abschnitt 3.5 vorgestellten jLAF-Framework steht bereits ein mächtiges Werkzeug zur Verwaltung und Generierung von HTML-Webinhalten zur Verfügung. Deshalb bietet es sich an,

das Framework um eine RIA-Komponente zu erweitern. Somit wird in diesem Kapitel eine serverseitige Lösung in Form einer Erweiterung des jLAF vorgestellt sowie auch ein clientseitiges Framework, mit Hilfe dessen sich derartige RIAs gestalten und verwalten lassen. Die Praxistauglichkeit der Kombination von jLAF auf der Serverseite und dem in diesem Kapitel eingeführten Framework auf der Clientseite wird am Ende dieses Kapitels in Form des Praxisbeispiels *Webbuch* unter Beweis gestellt.

5.1.1 Statische vs. dynamische RIAs

Um im weiteren Verlauf dieser Arbeit eine Unterscheidung zwischen "vollwertigen" RIAs (also tatsächlichen Web-*Anwendungen*) und Webauftritten, die nur aus Gründen der vielfältigen Gestaltungsmöglichkeiten mit Hilfe von RIA-Technologie erstellt wurden, ausdrücken zu können, soll den weiteren Ausführungen zunächst eine Definition vorangestellt werden:

Definition: Statische RIA

> Eine *statische RIA* ist eine Webanwendung, welche mit multimedialen Elementen und einer reichhaltigen Benutzerführung aufwartet, ihrem Charakter nach aber keine Applikation, sondern das Pendant zu einer inhaltszentrierten Webseite[1] darstellt.

Der Inhalt statischer RIAs entsteht demnach nicht durch eine unmittelbare Interaktion mit dem Benutzer, sondern liegt bereits beim Aufruf der RIA vor. Dabei soll es keine Rolle spielen, *wie lange* die Informationen bereits vorliegen. Wichtig ist nur, dass sie unabhängig von Benutzereingaben sind und somit bereits beim Aufruf der RIA bekannt sind. Um tatsächliche Anwendungen handelt es sich bei statischen RIAs demnach nicht.

Beispiele: Ein gängiges Beispiel für statische RIAs ist zum Beispiel ein Online-Prospekt eines Herstellers. Die Produktbeschreibungen, die technischen Daten und nicht zuletzt die Preisangaben und Bezugsquellen liegen unabhängig von Benutzereingaben und mithin bereits beim Aufruf der Seiten bzw. RIA vor.

Das Gegenstück zu statischen RIAs bilden dynamische RIAs. Auch diese sollen zunächst formal definiert werden:

Definition: Dynamische RIA

> Eine *dynamische RIA* ist eine RIA, deren darzustellende Inhalte überwiegend aufgrund einer Interaktion mit dem Benutzer gewonnen (z.B. errechnet oder generiert) werden.

Somit stellen dynamische RIAs das RIA-Pendant zu interaktionszentrierten Webseiten (vgl. Definition auf Seite 79) dar. Dynamische RIAs werden Gegenstand des nächsten Kapitels (Kapitel 6) sein. Die Ausführungen des aktuellen Kapitels beziehen sich indes ausschließlich auf statische RIAs.

[1] Die Bezeichnung "inhaltszentrierte Webseite" bezieht sich dabei auf die Definition von Seite 79.

5.1.2 HTML-Fallback-Lösungen und individualisierbare RIAs

Webauftritte, die ihrem Charakter nach zu den statische RIAs gezählt werden können, lassen sich in zunehmender Zahl im World Wide Web ausmachen. Der Trend hin zu dieser Form von Webauftritten lässt sich leicht an der Tatsache erkennen, dass viele informierende Webseiten, die in jüngster Zeit eine Überarbeitung erfahren haben, eben nun als statische RIA vorliegen. Dies gilt insbesondere für Nachrichtenportale. Ein Beispiel dafür ist das Nachrichtenportal des ZDF[2]: seit seinem Re-Design im Frühjahr 2009 setzt es verstärkt auf multimediale Inhalte, d.h. auf Ton- und/oder Videobeiträge sowie auf eine interaktive Präsentation von "trockenen" Informationen, wie beispielsweise Umfragedaten, Wahlergebnissen oder ähnlichen Datenanalysen.



Abbildung 5.1: Das ZDF-Politbarometer: Ein Beispiel für eine statische RIA

Ein Beispiel für die Integration einer RIA-Komponente in den Webauftritt des ZDF-Nachrichtenportals `heute.de` zeigt der Screenshot in Abbildung 5.1. Der Besucher der Seite kann sich in reichhaltiger, interaktiver Form über aktuelle Wahlprognosen informieren, die dargestellten Informationen liegen aber bereits fest vor (statische RIA). Die Prognosewerte könnten ebensogut auch als vergleichsweise "trockene" tabellarische HTML-Darstellung gezeigt werden, ohne an Informationsgehalt zu verlieren. Gerade die attraktive Gestaltung der Grafiken soll aber den Benutzer ansprechen, sodass das ZDF an dieser Stelle zu einer RIA-Technologie greift.

Die Neugestaltung des Webauftritts von `heute.de` ist aber auch in anderer Hinsicht ein Exempel für die Klasse der statischen RIAs: in den entsprechenden Foren des ZDF wurde höchst kontrovers diskutiert, ob und inwieweit der neue Auftritt barrierefrei ist und von Menschen benutzt werden kann, die nicht über den Flash-Player verfügen[3]. Wie die Ausführungen des einführenden Kapitels über RIAs gezeigt haben (vgl. dazu insbesondere Abschnitt 4.5), ist die Fokussierung auf Plattformen wie zum Beispiel den Flash-Player in Bezug auf Barriere-

[2] zu erreichen unter der URL `www.heute.de`

[3] Die multimedialen Elemente auf `heute.de` setzen stets die Existenz des Flash-Players auf dem Zielrechner voraus.

freiheit nicht unproblematisch. Ebenso darf die Frage, wie die Inhalte zugänglich sind, wenn die RIA-Zielplattform auf dem Ausgabegerät nicht verfügbar ist (z.B. Mobiltelefone), nicht unberücksichtigt bleiben.

5.1.2.1 HTML-Fallback-Lösungen

Eine naheliegende Lösung für diese Problemstellungen ist das Bereithalten der Inhalte in zwei (oder mehr) verschiedenen Versionen:

1. als (primär intendierte) RIA-Version für die jeweilige Zielplattform sowie
2. als HTML-Version, die zwar die Inhalte wiedergibt, auf multimediale und interaktive Komponenten jedoch verzichtet.

Für die "abgespeckte" HTML-Version des Webauftritts soll zunächst ein Begriff eingeführt werden:

Definition: HTML-Fallback-Lösung

> Eine *HTML-Fallback-Lösung* einer statischen RIA ist eine inhaltlich zwar gleichwertige, jedoch auf die standardkonforme Verwendung von HTML und CSS reduzierte Version der ursprünglichen RIA.

An dieser Stelle wird deutlich, weshalb das vorliegende Kapitel lediglich *statische* RIAs betrachtet: nur sie erlauben die automatische Generierung einer "inhaltlich gleichwertigen" HTML- und CSS-Repräsentierung. Zwar lassen sich – wie insbesondere aus der Diskussion der JavaScript-Frameworks (vgl. Abschnitt 4.2.1 auf Seite 84) hervorgeht – selbstverständlich auch vollwertige Applikationen mit HTML, CSS und JavaScript realisieren, sodass sie als Fallback-Version von dynamischen RIAs zur Verfügung stehen können, jedoch wird das Ziel in diesem Kapitel sein, Fallback-Versionen *automatisiert* zu gewinnen, d.h. generieren zu lassen. Dies ist jedoch nur bei statischen RIAs möglich.

5.1.2.2 Trennung von Layout und Inhalt bei statischen RIAs

Damit sich eine HTML-Fallback-Version generieren lässt, muss bei der Entwicklung der RIA abermals eine sehr sorgfältige Trennung zwischen Layout und Inhalt vorgenommen werden. Das praktische Vorgehen hierzu ist ein völlig anderes, als es die Trennung von Layout und Inhalt bei HTML-basierten Seiten erfordert. Dem beschreibenden, wohlstrukturierten Charakter von HTML-Dokumenten steht auf der RIA-Seite ein völlig ins Belieben des Entwicklers gestellter Aufbau der RIA gegenüber, welcher zudem zum Zeitpunkt der Anzeige nicht mehr im Quelltext, sondern compiliert vorliegt und eine Analyse der Struktur umso schwerer, wenn nicht sogar unmöglich macht.

Um dennoch eine sorgfältige Trennung von Darstellung und Inhalt zu erreichen, empfiehlt es sich, die darzustellenden Inhalte erst zur Laufzeit in die jeweiligen Positionen auf der Seite einzubetten. Mit anderen Worten: zur Compilierzeit der RIA sind sie noch nicht bekannt, sondern

sie werden erst zur Laufzeit aus einer separaten Quelle (z.B. Dateisystem oder Datenbank) geholt. Ein Beispiel dafür, wie (einfach) dies im Flex-Framework[4] möglich ist, zeigt Listing 5.1.

Listing 5.1: Beispiel für das Laden von externen Inhalten in eine Flex-RIA

```
1  var loader: URLLoader = new URLLoader(); // object for handling http request
2
3  private function getContent() : void {
4    var req: URLRequest = new URLRequest("http://www.myData.com/data.xml");
5    loader.addEventListener(Event.COMPLETE, onLoad);
6    loader.addEventListener(IOErrorEvent.IO_ERROR, onErr)
7    loader.load(req);
8  }
9
10 private function onLoad(ev: Event) : void {
11   txtContent.text = loader.data; // txtContent is Text-Area-GUI-Widget
12 }
13
14 private function onErr(ev: ErrorEvent) : void {
15   // handle error event
16 }
```

Erklärung zum Beispiel: Die Methode `getContent()` aus Listing 5.1 zeigt, wie in Flex (bzw. in der zugrundeliegenden Programmiersprache *ActionScript*) Inhalte aus externen Quellen geladen werden. Dem Konstruktor für ein URL-Request-Objekt wird als String übergeben, welche Datei geladen werden soll (Zeile 4). Ebensogut wäre auch ein Zugriff auf Dateien im lokalen Dateisystem möglich. Für diesen Fall wird statt der URL der Pfad zur lokalen Datei (ohne Protokoll-Prefix) notiert. Das Holen von Daten aus externen Quellen geschieht in Flex bevorzugt asynchron: anstatt auf den Abschluss einer Leseoperation zu warten, steht die Anwendung sofort wieder zur Verfügung und erfüllt die Leseaufgabe im Hintergrund. Durch entsprechende Ereignisse wird der Anwendung dann signalisiert, dass die Leseoperation (erfolgreich oder nicht erfolgreich) beendet wurde, sodass die Anwendung dann mit Hilfe von Ereignisbehandlern geeignet reagieren kann. In den Zeilen 5 und 6 werden zwei Ereignisbehandler installiert: Zeile 5 spezifiziert die Methode, die aufgerufen wird, wenn die Leseoperation erfolgreich durchgeführt werden konnte und das Ergebnis der Anfrage vorliegt (dies ist hier die Methode `onLoad`). Zeile 6 setzt den Ereignisbehandler für einen Misserfolg beim Lesen. Der eigentliche Start der Anfrage erfolgt in Zeile 7. Die "Erfolgsbehandlungs"-Methode `onLoad` (Zeilen 9 bis 11) füllt ein Element des User Interfaces (hier: einen Text-Bereich) mit dem aus der externen Quelle erhaltenen Inhalt. Dieser ist in dem Feld `data` des (global bekannten) `URLLoader`-Objektes enthalten (Zeile 10).

An dieser Stelle sei noch einmal darauf hingewiesen, dass das Vorgehen, zur Laufzeit der RIA Inhalte aus anderen Quellen (Server) zu beziehen, keinen Widerspruch zu den in diesem Kapitel betrachteten *statischen* RIAs bedeutet. Das Attribut *statisch* bezeichnet hier nur die Tatsache, dass die Inhalte nicht erst aufgrund einer Interaktion mit dem Benutzer entstehen (z.B. berechnet werden), sondern unabhängig von der RIA bereits anderweitig entstanden sind. Im betrachteten Beispiel bedeutet das konkret, dass der Inhalt der zu ladenden Datei `data.xml` unabhängig von der RIA erstellt wurde. Die enthaltenen Daten sind also *keine* Reaktion auf eine Eingabe des Benutzers der RIA.

[4] An dieser Stelle sei noch einmal darauf hingewiesen, dass das Flex-Framework als technische Grundlage für die in dieser Arbeit entstandenen Lösungen verwendet wurde.

5.1.2.3 Individualisierbarkeit und Multilingualität

Neben der höchst "groben" Aufteilung in eine RIA- und eine HTML-Version eines Web-Auftritts sind durchaus feingranularere Abstufungen auch innerhalb einer jeweiligen Klasse vorstellbar. In den vorausgegangenen Kapiteln wurde bereits beschrieben, wie HTML- und CSS-basierte Webauftritte eine Anpassung an individuelle Bedürfnisse und Präferenzen ermöglichen. Dieses Kapitel erörtert dieselbe Frage im Kontext von (statischen) RIAs.

Eine strikte Trennung von Layout und Inhalt bei RIAs, kombiniert mit dem Reichtum an Gestaltungsfreiheit und gewinnbringend einsetzbarer *User Experience*, den RIAs ihrer "Natur" nach anzubieten in der Lage sind, kann selbstverständlich auch dazu verwendet werden, dem Benutzer weitgehende Möglichkeiten zu offerieren, (s)eine RIA nach seinen eigenen Wünschen und Anforderungen optisch anzupassen. So kann eine RIA beispielsweise dem Benutzer erlauben, Variationen des Layouts zu wählen und dazu beispielsweise neben dem Standard-Layout auch eines mit besonders gut lesbaren Schriften und eines mit besonders hohen Kontrasten zur Auswahl stellen. Neben der rein optischen Komponente wäre es aber schön, wenn sich zudem auch eine sprachliche Anpassung der RIA (d.h. eine Unterstützung von Mehrsprachigkeit) erreichen ließe. Allgemeiner formuliert bedeutet dies, dass die vielfältigen Möglichkeiten einer RIA dazu verwendet werden sollen, dass der Benutzer die RIA "individualisieren" kann.

Zwar ist der Gedanke, dass Anwendungen individualisiert, lokalisiert und in unterschiedlichen Sprachversionen vorgehalten werden können, kein neuer und kein ungewöhnlicher. Demzufolge existieren dazu etablierte Lösungen, insbesondere auch solche zur Unterstützung von Mehrsprachigkeit in (Web-)Applikationen (siehe z.B. [Ber04] für den Umgang mit Lokalisierungen in JavaServerPages und Servlets). Auch das hier verwendete Flex-Framework wartet mit einem eigenen Ansatz zu diesem Problemfeld auf (vgl. dazu auch Abschnitt 5.3.3.1). Dennoch schöpft dieser – wie später noch gezeigt wird – nicht das ganze Potenzial aus, das RIAs bieten.

5.1.3 Zieldefinition

Motiviert durch die vorausgegangenen Ausführungen wird im restlichen Teil des vorliegenden Kapitels nun ein Vorgehen gezeigt, wie sich eine statische RIA dergestalt konzipieren lässt, dass sie

1. eine automatisierte Generierung von barrierefreien HTML-Fallback-Lösungen erlaubt,
2. eine individualisierbare Darstellung innerhalb der RIA ermöglicht und dabei insbesondere auch
3. Mehrsprachigkeit unterstützt.

Die Erklärung des dabei präsentierten Lösungsansatzes erfolgt getrennt nach Clientseite (RIA) und Serverseite (*"Backend"*). Die gezeigten Konzepte sind jedoch stets als Einheit zu sehen, d.h. die serverseitige Lösung bedingt den Einsatz der beschriebenen clientseitigen Lösung und umgekehrt.

5.1.3.1 Integration mit jLAF

Betrachtet man den ersten Punkt der zuvor genannten Aufzählung, so impliziert dieser ein Ziel, das bereits in Zusammenhang mit dem WebTool formuliert wurde: die Generierung bar-

rierefreier HTML-Seiten. Zur Erreichung dieses Ziels wurde in Kapitel 3 zunächst das WebTool und daraus dann schließlich das jLAF-Framework gewonnen.

Infolgedessen liegt mit dem jLAF-Framework bereits ein Fundament vor, um das soeben formulierte Ziel einfach erreichen zu können. Dadurch reduziert sich die aktuelle Problemstellung auf die Frage, wie sich das jLAF-Framework in eine RIA-basierte Umgebung[5] integrieren lässt. Gleichbedeutend damit lässt sich die Frage auch wie folgt formulieren: können statische RIAs vom jLAF-Framework profitieren?

5.1.3.2 Eine Bemerkung zur verwendeten RIA-Technologie

Die clientseitig beschriebenen Lösungsansätze sind unabhängig von einer bestimmten RIA-Technologie verwendbar, adressieren jedoch RIAs der Kategorie 2[6]. Im Rahmen der Implementierung des Praxisbeispiels "*Webbuch*" (siehe Abschnitt 5.5 auf Seite 135ff.) wurde das Flex-Framework von Adobe verwendet. Deshalb sind die clientseitig zitierten Code-Ausschnitte in der Programmiersprache ActionScript bzw. in MXML geschrieben. Dies soll jedoch keinesfalls suggerieren, dass die beschriebenen Ideen auf einen Einsatz mit dem Flex-Framework beschränkt sind.

5.2 Lösungsansatz auf der Serverseite

5.2.1 Das Direktivenkonzept im jLAF-Framework

In Zusammenhang mit der Beschreibung des jLAF-Frameworks wurde bereits kurz das Direktivenkonzept erläutert (vgl. Abschnitt 3.5.5 auf Seite 72). Eine Direktive war dabei als ein Methodenaufruf, der anstelle des tatsächlichen Inhalts im Seitenquellcode steht, eingeführt worden. Wenn das jLAF-Framework während der Generierung der Seiten auf Direktiven trifft, wird via Reflection die genannte Methode aufgerufen. Ihr Rückgabewert (welcher stets vom Typ `String` sein muss) ersetzt nach erfolgreichem Durchlauf der Methode die Direktive in der generierten Seite. Für Methoden, die zum Aufruf als Direktiven bereitstehen, wurde im Kapitel über das jLAF-Framework der Begriff *Remote Access Points (RAPs)* verwendet.

Bereits während der Ausführungen zum jLAF in Kapitel 3.5 wurde bereits in Aussicht gestellt, dass das Konzept der Remote Access Points neben der Verwendung als Direktiven auch in anderer Hinsicht von Bedeutung sein würde. Dies nahm den im folgenden konkretisierten Gedanken vorweg, dass auch Rich Internet Applications *zur Laufzeit* diese RAPs aufrufen und ihre Rückgabestrings weiterverarbeiten können. Somit kommt RAPs eine Doppelbedeutung zu: sie liefern Inhalte für

- mit Hilfe des jLAF-Frameworks generierte HTML-Seiten sowie
- Rich Internet Applications (hier auch zur Laufzeit!).

Mithin ist also ein für dieses Kapitel formuliertes Teilziel – eine automatisierte Generierung einer HTML-Fallback-Version von statischen Rich Internet Applications – bereits erreicht: die

[5] Hier sei nochmals darauf aufmerksam gemacht, dass *statische* RIAs betrachtet werden.

[6] vgl. Seite 83 für eine Definition der Kategorien sowie die Abschnitte 4.2.2ff. auf Seite 85ff. für eine Vorstellung der RIA-Technologien

Verwendung des jLAF zur Erzeugung der HTML-Seiten sowie darüberhinaus der Zugriff auf die Remote Access Points aus dem RIA-Client heraus ermöglichen, neben der RIA-Version auch über eine HTML-Fallback-Lösung verfügen zu können, wobei letztere ohne weitere manuelle Eingriffe gewonnen – mithin also automatisch generiert – werden kann. Einmalig müssen das Layout erstellt sowie die Konfiguration des jLAF vorgenommen werden, danach kann die HTML-Version der statischen RIA durch einen einfachen Generierungslauf synchron zu inhaltlichen Änderungen in der RIA gehalten werden.

5.2.2 Das RAP-Pattern und Rich Internet Applications

Das Ziel des RAP-Patterns ist es, einen flexiblen und vollständig gekapselten Zugriff auf die Inhalte (Texte und Daten) zu haben, aus denen Webseiten (sowohl in RIAs als auch in HTML-Versionen) generiert werden sollen. Dementsprechend kann durch die Anwendung des RAP-Patterns erreicht werden, dass die Inhalte als HTML-Repräsentierung, in Form eines XML-Dokuments oder auch unmittelbar als Datenobjekte (*Data Access Objects, DAOs*) geliefert werden. Ferner abstrahiert das RAP-Pattern den Zugriff auf die Methoden zur Erzeugung der gewünschten Inhaltsausprägungen: durch das Implementieren entsprechender Schnittstellenmethoden bieten die beteiligten Klassen ihre Repräsentierung als HMTL- und/oder XML-Dokument an. Ferner lassen sich die Repräsentierungen ggf. auch direkt als Objekt-Auflistung (`Collection`) erhalten. RAPs lassen sich im jLAF mit jeweils nur einer Zeile Programmcode pro neu aufzunehmendem RAP sehr einfach hinzufügen.

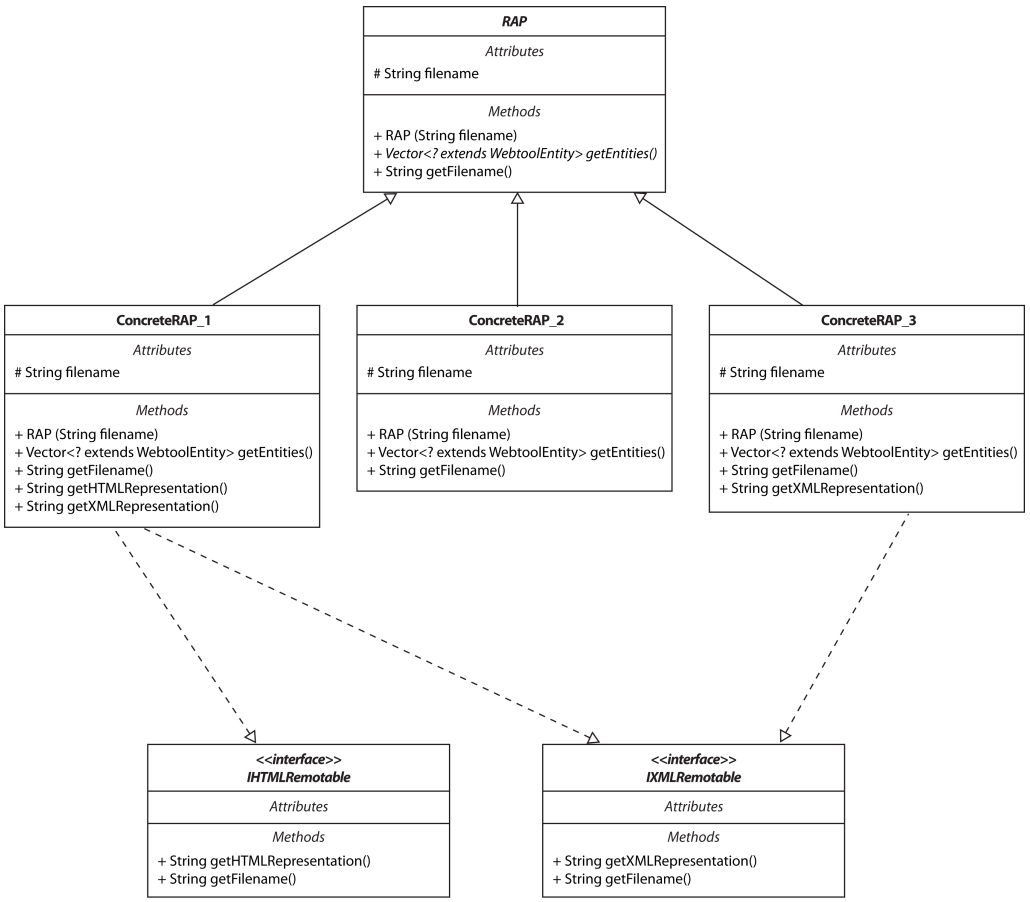


Abbildung 5.2: High-Level-Überblick über das RAP-Pattern

Abbildung 5.2 gibt in Form eines UML-Klassendiagramms einen *High Level*-Überblick über die Struktur des RAP-Patterns. Es sieht als zentrale, abstrakte Klasse die Klasse `RAP` vor, welche zunächst unabhängig von einer Verwendung zur HTML-, XML- oder DAO-Übertragung implementiert werden muss. In Listing 5.2 ist der Inhalt der abstrakten Klassen `RAP` aufgeführt. Eine Implementierung dieser Klasse bezieht sich stets auf eine bestimmte Entity-Klasse (siehe Abschnitt 3.5.4.1 auf Seite 66). Gibt es beispielsweise eine Entity-Klasse namens `GuestbookEntryEntity` zur Repräsentierung von Gästebuch-Einträgen, so wird eine zugehörige RAP-Implementierung (z.B. `GuestbookEntryRAP`) darauf spezialisiert sein, einen entfernten Zugriff (nur) auf diese Entity-Objekte anzubieten. Jede RAP-Implementierung muss mindestens die abstrakte öffentliche Methode `getEntities()`, welche die jeweiligen Entity-Objekte als Vektor zurückliefert, enthalten. Im einfachsten Fall besitzt eine Implementierung dieser Klasse tatsächlich *nur* die Methode `getEntities()`. Derartige Implementierungen eignen sich zum Einsatz im Rahmen der *Remote Method Invocation*. Sollen darüberhinaus die Daten (d.h. die Entities) auch als XML- und/oder HTML-Dokumente aufbereitet werden können, um z.B. einen Zugriff via Servlet zu unterstützen, so können die konkreten Ausprägungen der Klasse `RAP` zudem auch die Interfaces `IHTMLRemotable` und `IXMLRemotable` implementieren. Dann stellen sie neben der ohnehin vorhandenen Methode `getEntities()` noch die Methode(n) `getXMLRepresentation()` bzw. `getHTMLRepresentation()` zur Verfügung, welche – wie ihre Bezeichnung bereits anzeigt – anstelle der DAO-Collections entsprechende XML- oder HTML-Dokumente zurückgeben. Auch dies ist in Abbildung 5.2 skizziert: die RAP-Klasse `ConcreteRAP_2` implementiert keine Interfaces und steht somit nur für den Gebrauch in einem Remoting-Szenario zur Verfügung. Die übrigen `ConcreteRAP`-Klassen implementieren zusätzlich noch ein weiteres (bzw. zwei) Interface(s).

Listing 5.2: Die abstrakte Klasse `RAP`

```
1  public abstract class RAP {
2
3    protected String filename;
4
5    public RAP(String filename) {
6      this.filename = filename;
7    }
8
9    public abstract Vector<? extends WebtoolEntity> getEntries();
10
11   public String getFilename() {
12     return filename;
13   }
14 }
```

Beispiel: Im Vorgriff auf das in Abschnitt 5.5 vorgestellte Praxisbeispiel soll bereits an dieser Stelle mit Listing 5.3 ein Beispiel gezeigt werden, wie eine Implementierung der vom Interface `IHTMLRemotable` geforderten Methode `getHTMLRepresentation()` aussehen kann.

Listing 5.3: Beispielimplementierung einer Methode zum Aufbau einer HTML-Repräsentierung

```
1  public class GuestbookEntryRAP extends RAP implements IHTMLRemotable {
2
3    // ...
4
5    public String getHTMLRepresentation() throws RAPException {
6      StringBuilder sb = new StringBuilder();
7      Vector entries = getEntries();
8      if (entries == null) {
```

```
9         throw new RAPException("Could not build Guestbook entries");
10      }
11      sb.append("<div id=\"guestbook\">");
12      for (Object entry : entries) {
13        sb.append(((GuestbookEntryEntity) entry).getHTMLRepresentation());
14      }
15      sb.append("</div>"); // div guestbook
16      return sb.toString();
17    }
18  }
```

Erläuterungen zu Listing 5.3: Die Methode dient zur HTML-Aufbereitung von eingegangenen Gästebucheinträgen. Jeder Eintrag wird intern als Instanz der Klasse `GuestbookEntryEntity` (einer Implementierung der abstrakten Klasse `WebtoolEntity`) repräsentiert. In Zeile 3 des Codebeispiels wird durch den Aufruf der Methode `getEntities()`[7] ein Vektor erstellt, welcher alle Einträge jeweils als Entity-Objekt enthält. Die Zeilen 7 bis 12 bauen dann das HTML-Dokument bzw. -fragment auf, indem für jedes Eintrag-Objekt (vgl. Typumwandlung in Zeile 9) die Methode `getHTMLRepresentation()` aufgerufen wird. Letztendlich bedeutet dies, dass hier Funktionalität des jLAF unmittelbar verwendet werden kann, denn der Aufruf entsprechender Methoden von Entity-Objekten ist bereits elementarer Gegenstand der Funktionsweise des jLAF-Frameworks.

Wird das jLAF lediglich zur Verwaltung von HTML-Seiten verwendet, agiert es als lokale Applikation. Die Frage der Erreichbarkeit der als Direktiven verwendeten Methoden (RAPs) "von außen" stellt sich in diesem Fall nicht. Anders jedoch ist dies, wenn die RAPs auch von RIAs kontaktiert werden sollen, denn dann benötigt das Framework einen "Einstiegspunkt", über den die RIAs auf die RAPs zugreifen können. (Man beachte an dieser Stelle, dass in diesem Fall die RIA als Client und das jLAF-Framework als Server agieren.)

Zur Erreichbarkeit der RAPs sind die folgenden Möglichkeiten vorhanden:

Servlet-basierte RAPs: Die RAPs werden über traditionelle Servlets angesprochen, d.h. die RIA ruft ein Servlet auf, welches dann als Vermittler zum RAP auftritt. Dieses Vorgehen setzt einen Applikationsserver (Servlet-Container) voraus, der von der RIA zur Laufzeit kontaktiert wird.

Remoting Mechanismen: Eine besonders elegante und flexible Art, auf RAPs zuzugreifen, liegt in Form von Remoting-Mechanismen vor. Hier erfolgt der entfernte Aufruf in Form eine *Remote Method Invocation*, d.h. die RIA kommuniziert mit den RAPs unmittelbar.

Vorab generierte RAP-Ausgaben: die von der RIA anzuzeigenden Inhalte werden nicht in Echtzeit durch das jLAF bereitgestellt, sondern "vorproduziert". Sie liegen also statisch auf dem Server. Diese Möglichkeit ist (nur) dann sinnvoll, wenn zur Auslieferung des RIA-Clients auf den Einsatz eines Applikations-Servers verzichtet und die RIA stattdessen nur von einem Web-Server ausgeliefert werden soll.

Während die letzte Möglichkeit lediglich der Vollständigkeit halber hier erwähnt wurde, erlauben die ersten beiden Möglichkeiten sehr flexible, den jeweiligen Gegebenheiten anpassbare Lösungen. Sie werden nachfolgend in jeweiligen Unterabschnitten genauer erläutert.

[7] Weil konkrete RAP-Klassen stets Implementierungen der abstrakten Klasse `RAP` sind, ist die Methode `getEntities()` in jedem Fall vorhanden.

5.2.3 Servlet-basierte RAPs

Der Aufruf der RAPs via Servlets ist trivial: ein Servlet dient lediglich als *Wrapper* für einen RAP-Aufruf. Die Parameter, die dem Servlet übergeben wurden, werden an den RAP weitergereicht, und ebenso liefert das Servlet die erhaltene Rückgabe des RAPs weiter an den Aufrufer zurück. Der in diesem Zusammmenhang relevante Code eines Servlets ist in Listing 5.4 beispielhaft gezeigt:

Listing 5.4: Zugriff auf RAPs über den "Umweg" eines Servlets

```
1  protected void doGet(HttpServletRequest request, HttpServletResponse response)
2    throws ServletException, IOException {
3      processRequest(request, response);
4  }
5
6  protected void doPost(HttpServletRequest request, HttpServletResponse response)
7    throws ServletException, IOException {
8      processRequest(request, response);
9  }
10
11 protected void processRequest(HttpServletRequest request, HttpServletResponse
       response)
12   throws ServletException, IOException {
13     response.setContentType("text/html;charset=UTF-8");
14     PrintWriter out = response.getWriter();
15     String filename = request.getParameter("filename");
16     out.println(new WebbookRAPs().getGuestbookEntriesInHTML(filename));
17     out.close();
18   }
19 }
```

Erläuterungen zum Listing 5.4: Das Codebeispiel zeigt einen Ausschnitt aus einem gewöhnlichen Standard-Servlet und entstammt dem noch folgenden Praxisbeispiel (siehe Abschnitt 5.5). Die bei einem Servlet relevanten Methoden `doGet` und `doPost` delegieren ihre "Aufgabe" an die Methode `processRequest` weiter. Letztere konfiguriert die Ausgabe (Zeile 13) und initialisiert den Ausgabestrom (Zeile 14). In diesen Ausgabestrom wird der Rückgabewert der RAP-Methode `getGuestbookEntriesInHTML` geschrieben (Zeile 16). Ein Schließen des Ausgabestroms beendet den Schreibprozess (Zeile 17).

Mit Ausnahme einer Zeile (Zeile 16) zeigt Listing 5.4 nur Code, der bei *jedem* Servlet – also insbesondere auch solchen ohne jLAF-Bezug – so oder in äquivalenter Form notiert werden muss. Das bedeutet somit aber auch, dass sich servletbasierte RAPs mit nur einer einzigen Zeile Code in Servlets integrieren lassen, sodass die Verwendung des RAP-Patterns auch einen entscheidenden Beitrag zur Vereinfachung der Entwicklung und zur Verbesserung der Codequalität leistet.

Via Servlets lassen sich nur Zeichenketten (*Strings*) zurückliefern. Wenn komplexe Datenstrukturen via Servlets transportiert werden sollen, dann müssen diese Strukturen in irgendeiner Form serialisiert werden. In der Regel wird dies die Repräsentierung der Datenstruktur als XML-Dokument oder in Form der JSON-Notation[8] bedeuten. Der Austausch von Datenstrukturen als XML- oder JSON-Dokument ist sehr flexibel und stellt keine hohen Anforderungen

[8] JSON (= *JavaScript Object Notation*) ist ein Format zum Datenaustausch, das – entgegen dessen, was die Benennung vermuten lässt – nicht nur auf die Verwendung mit JavaScript beschränkt ist, sondern auch von vielen weiteren Programmiersprachen unterstützt wird.

an die Integration zwischen Aufrufer und Servlet: der Aufrufer muss lediglich in der Lage sein, XML-Dokumente (respektive JSON-Dokumente) zu verarbeiten. Weitere Ansprüche an eine Kompatibilität mit dem Server sind nicht gegeben.

Diese Form des Datenaustauschs ist mithin sehr universell einsetzbar, man "erkauft" sich diese Universalität jedoch durch die Notwendigkeit, die Datenstrukturen auf der Serverseite vor dem Versenden in ein XML- bzw. JSON-Dokument zu überführen und dieses auf der Client-Seite wieder zu deserialisieren, d.h. dort die empfangenen Daten in eine weiterverarbeitbare Struktur zu bringen. Eine wesentlich elegantere Form, Daten zwischen Client (hier: RIA) und Server (hier: RAP) auszutauschen, ist die Verwendung von sogenannten *Remoting* Mechanismen.

5.2.4 RAPs und Remoting Mechansimen

Durch die Verwendung von Remoting Mechanismen kann auf das aktive (d.h. durch den Programmierer bewusst vorgenommene) Serialisieren und Deserialisieren von Daten, die zwischen Client und Server versendet werden sollen, verzichtet werden. Stattdessen tauschen Client und Server direkt *Objekte* aus. Letztendlich wird "hinter den Kulissen" zwar auch eine Serialisierung bzw. Deserialisierung vorgenommen, jedoch fällt dies beim Einsatz eines Remoting Mechanismus nicht mehr in die Verantwortung des Programmierers, sondern wird eben von diesem Mechanismus übernommen.

Neben dem Austausch von Objekten umfasst das *Remoting* aber auch das Aufrufen von Methoden an "entfernter" Stelle (*Remote Procedure Calls, RPCs)* bzw. – im Java-Kontext – *Remote Method Invocation (RMI).* Im Kontext dieses Abschnitts bedeutet *Remoting* deshalb, dass

- der RIA-Client die RAP-Methoden auf dem Server *unmittelbar* (also nicht durch einen "Umweg" über Servlets o.ä.) aufrufen kann und
- der RIA-Client mit dem Server Objekte (DAOs) anstelle von serialisierten Daten (z.B. XML-Dokumenten) austauscht.

Im Unterschied zur XML-Lösung bei Servlet-basierten RAPs setzt der Einsatz eines Remoting-Mechanismus voraus, dass eine solche Kommunikation zwischen Client- und Servertechnologie überhaupt möglich ist. Im Falle des für diese Arbeit verwendeten Flex-Frameworks steht für das *Remoting* die Open-Source-Lösung "BlazeDS" zur Verfügung, welche für die Praxisbeispiele dieser Arbeit im folgenden nun verwendet wird. Weil eine detaillierte Erkärung von BlazeDS den Rahmen dieser Arbeit sprengen würde und auch nicht unmittelbarer Gegenstand dieser ist, sei für eine Einführung in die Remoting-Fähigkeiten von BlazeDS auf [Mü09] oder die zahlreichen Online-Dokumentationen verwiesen. Für das Verständnis der in diesem Abschnitt erläuterten Idee der "Remoting RAPs" sind tiefergehende Kenntnisse über BlazeDS nicht erforderlich. Desweiteren sei an dieser Stelle ausdrücklich darauf hingewiesen, dass auch für andere RIA-Frameworks vergleichbare Lösungen bereits zur Verfügung stehen oder zumindest angekündigt sind (Stand: September 2009), sodass die im folgenden erläuterten Ideen keinesfalls nur auf den Einsatz mit einer bestimmten Technologie (namentlich: Flex) beschränkt sind.

Der Einsatz eines Remoting-Mechanismus in Zusammenhang mit RIAs ist primär darauf ausgelegt, eine Kommunikation des Clients (RIA) mit dem Backend-Server zu vereinfachen. In der Regel kommt das Remoting daher wohl bevorzugt in vollwertigen Applikationen (d.h. dynamischen RIAs gemäß der Definition auf Seite 106) zum Einsatz. Das vorliegende Kapitel

beschäftigt sich jedoch mit statischen RIAs, d.h. ein "reger" Austausch von Daten und Objekten zwischen Client und Server steht gar nicht so sehr im Mittelpunkt der Betrachtungen. Dennoch kann ein Remoting-Mechanismus in diesem Szenario sehr gewinnbringend eingesetzt werden, denn

- er erlaubt eine weitgehend nahtlose Kooperation zwischen (RIA-)Client und Backend-Server und
- er ermöglicht eine wesentlich schnellere Übertragung der Daten, weil zum Datenaustausch oftmals ein Binärformat verwendet wird. (Im Fall von "BlazeDS" ist dies das *Action Message Format*).

Damit Client und Server miteinander Objekte austauschen können, müssen *Mapping*-Regeln konfiguriert werden. Diese Regeln ordnen einer Klasse auf der Serverseite eine Klasse auf der Client-Seite zu (bzw. umgekehrt). Nur, wenn diese Regeln vorliegen, verfügt der Client (resp. der Server) über die erforderlichen Kenntnisse, welcher eigenen Klasse das vom Server (resp. vom Client) erhaltene Objekt entspricht. Im Falle von BlazeDS obliegt dem Client die Angabe derartiger Regeln. Dieser Ansatz bringt den Vorteil, dass der Server keinerlei Kenntnisse über einen entfernten Client und erst recht nicht über Implementierungsdetails (Klassen) auf der Client-Seite besitzen muss. Jeder Client ist also dafür verantwortlich, die Mapping-Regeln zu erstellen und zu verwalten, der Server indes bleibt unverändert und hat keinerlei Kenntnisse über die sich mit ihm verbindenden Clients. Primär ist das Remoting also ein clientseitig zu verwendendes Konzept.

Somit kann das Konzept der *Remoting RAPs* sehr einfach in das jLAF integriert werden. Als Server agiert hier das Framework, das – wie zuvor erklärt – keinerlei Kenntnisse über den Client besitzen und insoweit also lediglich die RAP-Methoden öffentlich verfügbar machen muss. Empfehlenswert ist die Zusammenfassung der RAPs in einer eigenen Klasse, denn dann genügt es, diese Klasse öffentlich zugänglich zu machen. Sie stellt dann diejenigen Methoden zur Verfügung, die von außen erreicht werden sollen, der Rest des Frameworks bleibt jedoch von außen unerreichbar. Ein Codebeispiel für die konkrete Anwendung wird in Zusammenhang mit dem Praxisbeispiel in Abschnitt 5.5 gezeigt.

5.3 Lösungsansatz auf der (RIA-)Client-Seite

Wie erwähnt, ergänzen sich die server- und clientseitigen Konzepte des RAP-Patterns gegenseitig. Während im vergangenen Abschnitt die Serverseite betrachtet wurde, soll nun die Client-Seite im Fokus stehen, wobei hier mit *Client* fortan stets ein RIA-Client gemeint ist, der in Kategorie 2 der auf Seite 83 vorgenommen Klassifizierung einzuordnen ist. Es werden Lösungsmöglichkeiten gezeigt, um client-seitig

- den Bezug von Daten in einer separaten, austauschbaren Schicht realisieren zu können und
- technische Voraussetzungen für die Individualisierbarkeit des Clients zu schaffen, wobei dies insbesondere die Bereiche
  - Mehrsprachigkeit und
  - Layout-Anpassungen

betrifft.

Dabei sind die Lösungsmöglichkeiten so konzipiert, dass sie serverseitig das jLAF-Framework erwarten.

5.3.1 Bezug der Inhalte: die Client-Seite des RAP-Patterns

Wie im vorausgegangenen Abschnitt gezeigt wurde, kann die Kommunikation zwischen Server und RIA-Client wahlweise via Servlet oder via Remoting erfolgen. Beide Möglichkeiten sind – insbesondere beim verwendeten Flex-Framework – der Normalfall der Client-Server-Kommunikation.

In der Verwendung unterscheiden sich jedoch das Remoting und die Kommunikation via Servlet voneinander, hier insbesondere auch wieder in Flex. In der Konsequenz bedeutet dies, dass sich der Entwickler der RIA im Vorfeld der Programmierung bereits auf eine Kommunikationsart festlegen oder doppelten Programmieraufwand betreiben muss (falls *beide* Arten unterstützt werden sollen). Es fehlt hier bei Flex in der aktuellen Version an einer Abstraktion, welche einen flexiblen Austausch der Service-Schicht erlauben würde.

Aus diesem Grunde ist im Rahmen dieser Arbeit eine clientseitige Service-Abstraktion entwickelt worden, welcher der flexiblen Kommunikation, wie sie das jLAF durch das RAP-Konzept serverseitig bietet, eine ebenfalls flexible Kommunikationsabstraktion auf dem Client gegenüberstellt. Die entwickelte Lösung wird im folgenden als ein Pattern vorgestellt. Dieses Pattern erlaubt die vollständige Kapselung des Zugriffs auf den Server. Die Zugriffe erfolgen über eine eigene Schicht, sodass durch einen einfachen Austausch dieser Schicht ohne weitere Änderungen am Programm-Code die gesamte RIA auf einen anderen Zugriffsmechanismus umkonfiguriert werden kann.

Die beteiligten Klassen des clientseitigen RAP-Patterns zeigt das Klassendiagramm in Abbildung 5.3. Es macht deutlich, dass das clientseitige RAP-Pattern einer Hierarchie entspricht, an deren Spitze das Interface `IServiceConnector` steht. Der gesamte Programm-Code wird nur unter Verwendung dieses Interfaces erstellt, d.h. nirgends im Code wird mit einer tatsächlichen Implementierung gearbeitet, sodass eine einfache Austauschbarkeit der verwendeten Implementierung gewährleistet ist. Die Implementierung wird dann entweder an genau *einer* expliziten Stelle genannt oder auch gar nicht. Letzteres ist der Fall, wenn der Client beispielsweise mit Hilfe des Spring-Frameworks[9] bzw. einer vergleichbaren Lösung zum Einsatz kommen soll, denn dessen tragendes Konzept ist die sogenannte *Dependency Injection*. Dieses Konzept sieht vor, dass die Instanziierung von Klassen nicht im Code selber, sondern separat durch das Spring Framework erfolgt. Im Quellcode wird daher nirgends eine konkrete Implementierung genannt oder gar instanziiert. Stattdessen finden lediglich die Interface-Bezeichnungen Verwendung. Zur Laufzeit "injiziert" das Framework (dies erklärt die Benennung des Konzepts) die konkreten Implementierer, die in den Konfigurationsdateien des Frameworks (und nur da) genannt sind, in die Applikation.

Die Abstraktion des Zugriffsmechanismus' im vorliegenden Lösungsansatz der Client-Server-Kommunikation erlaubt es, dass nur in einer expliziten Konfigurationsdatei des RIA-Clients die tatsächlich gewünschte Implementierung genannt wird, der restliche Code aber ausschließlich mit Interfaces operiert. Aus diesem Grunde bleibt auch die Benennung des Interfaces insoweit unspezifisch, als sie nur von einem *Service*-Connector spricht und dessen konkrete Ausprägung durch die Namensgebung der implementierenden Klassen zum Ausdruck bringen lässt.

[9] siehe `http://www.springsource.org`

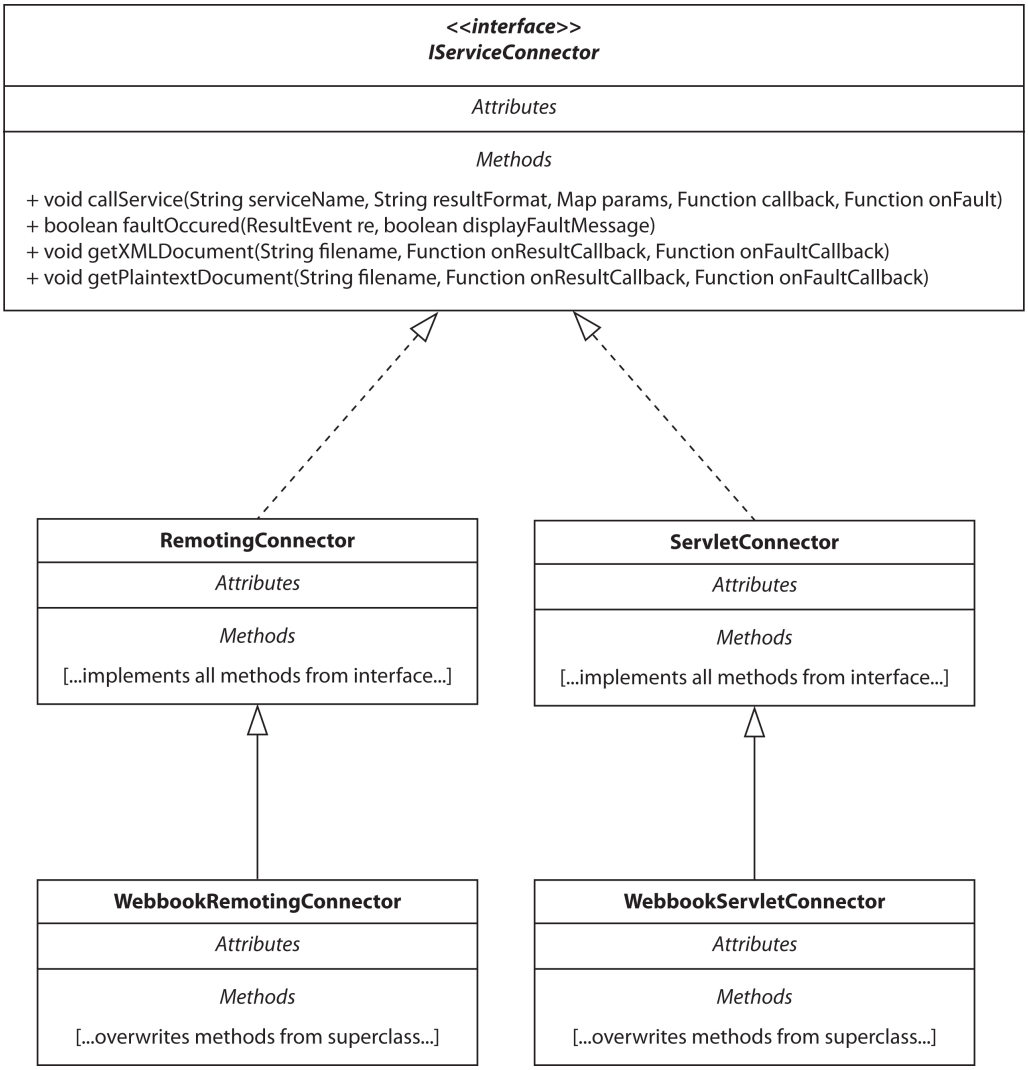


Abbildung 5.3: Kapselung des Zugriffs auf den Server via Interfaces

Als Implementierer des Interfaces `IServiceConnector` sind zunächst zwei abstrakte Klassen vorgesehen: `RemotingConnector` für den Zugriff via Remoting und `ServletConnector` für den Aufruf eines Servlets. Sie werden jeweils für einen bestimmten Bedarfsfall von einer abgeleiteten konkreten Klasse implementiert und zeigen – wie gerade erwähnt – durch ihre Benennung an, welchen Kommunikationskanal sie bedienen. Abbildung 5.3 zeigt in diesem Zusammenhang die konkreten Implementierer-Klassen `WebbookRemotingConnector` und `WebbookServletConnector`, welche auf eine Anwendung im Praxisbeispiel (siehe Abschnitt 5.5) zugeschnitten sind. Die existierenden Implementierungen und deren Ableitungen lassen sich selbstverständlich durch eigene Klassen (für evtl. weitere Kommunikationsarten) ergänzen.

Die zentrale Methode des Interfaces `IServiceConnector` ist die Methode `callService`. Die Parameter, mit denen diese Methode aufgerufen wird, haben die folgende Bedeutung:

`serviceName:` der Name des aufzurufenden Servlets bzw. der aufzurufenden Methode

`resultFormat:` das Format, in dem die Rückgabe erwartet wird; eine Bedeutung hat dieser Parameter nur, wenn er in Zusammenhang mit Servlets verwendet wird; bei der Verwendung des Remoting-Zugriffs ist dieser Parameter unbedeutend.

`params:` eine (assoziative) Liste von Parametern (Schlüssel-Werte-Paare), die dem Servlet bzw. der Methode beim Aufruf übergeben werden sollen

`callback:` die Methode, die aufgerufen werden soll, wenn der Zugriff auf das Servlet bzw. die entfernte Methode erfolgreich durchgeführt werden konnte und eine Antwort vorliegt

`onFault:` die Methode, die aufgerufen werden soll, wenn der Zugriff auf das Servlet bzw. die entfernte Methode *nicht* erfolgreich durchgeführt werden konnte; dieser Parameter ist optional: wird er nicht angegeben, so wird ersatzweise die Methode `onFault` des Implementierers von `IServiceConnector` verwendet.

Bei den letzten beiden Parametern handelt es sich um Ereignisbehandler (*Event Handler*). Durch ihre Verwendung ist unmittelbar erkennbar, dass ein Aufruf des Servlets bzw. der entfernten Methode stets asynchron erfolgt, d.h. der RIA-Client blockiert nicht für die Dauer der Bearbeitung der Anfrage auf dem Server, sodass mit ihm direkt weitergearbeitet werden kann. Über den Erfolg bzw. Misserfolg des entfernten Servlet- bzw. Methodenaufrufs wird der Client durch ein Ereignis informiert, auf das er mit den hier als Parameter übergebenen Ereignisbehandlern (Aufruf der Methoden) reagieren kann.

Die Implementierung der Methode `IServiceConnector` im "Servlet"-Fall ist trivial und entspricht dem Regelfall beim Zugriff auf HTTP-Services in Flex. Um eine Vergleichbarkeit zu der technisch deutlich diffizileren Variante auf der Seite der Remoting-Implementierung zu ermöglichen, ist sie – obwohl sie konzeptionell nichts Neues beiträgt – in Listing 5.5 gezeigt.

Listing 5.5: Implementierung der Methode `callService` für den Zugriff via Servlets

```
1  override public function callService(serviceName:String,
2      resultFormat:String, params:Object, callback:Function,
3      onFault:Function=null) : void {
4    var svc:HTTPService = new HTTPService();
5    svc.url = this.server + serviceName; // field "server" contains server name
6    svc.resultFormat = resultFormat;
7    svc.addEventListener(ResultEvent.RESULT, callback);
8    (onFault == null) ?
9      svc.addEventListener(FaultEvent.FAULT, onFaultDefault) :
10     svc.addEventListener(FaultEvent.FAULT, onFault);
11   svc.send(params);
12 }
```

Erläuterungen zu Listing 5.5: In Zeile 4 wird ein Objekt der Klasse `HTTPService` instanziiert, das einen Aufruf eines Servlets (via HTTP) initiiert und behandelt. In den Zeilen 5 bis 8 werden die relevanten Parameter für den Aufruf konfiguriert (Ziel-URL, erwartetes Rückgabeformat, Ereignisbehandlungsmethode im Erfolgs- sowie im Misserfolgsfall). Der Aufruf des Servlets erfolgt in Zeile 11. Wie erwähnt, geschieht die Anfrage asynchron, sodass beim Eintreffen der Antwort ein Aufruf der in den Zeilen 7 bzw. 9 oder 10 genannten Methoden erfolgt.

Die Implementierung der `callService`-Methode zur Verwendung mit Remoting-Mechanismen erfordert eine Abkehr von den in Flex üblichen Programmier-Gepflogenheiten zum Aufruf entfernter Methoden. In der Regel wird der Aufruf einer entfernten Methode in Flex genauso gehandhabt wie der einer "normalen" Methode, insbesondere also *synchron*. Dies würde der oben geschilderten Funktionsweise der durch das Interface `IServiceConnector` geforderten Methode `callService` zuwiderlaufen. Mit Hilfe von Reflection-Mechanismen kann – wie Listing 5.6 zeigt – die gewünschte Funktionsweise aber auch bei einem Zugriff via Remoting nachgebildet werden.

Listing 5.6: Implementierung der Methode `callService` für den Zugriff via Remoting

```
1  override public function callService(serviceName: String,
2      resultFormat: String, params: Object, callback: Function,
3      onFault: Function = null) : void {
4    var wro: RemoteObject;
5    (onFault == null) ?
6      wro = new RAPRemoteObject(callback, onDefaultFault) :
7      wro = new RAPRemoteObject(callback, onFault)
8    var op: AbstractOperation = wro.getOperation(serviceName);
9    var p: Array = new Array();
10   var j: int = 0;
11   for (var i:String in params) {
12     p[j] = params[i];
13     j++;
14   }
15   // ... call method via op.send(p) ...
16 }
```

Erläuterungen zu Listing 5.6: In den Zeilen 4 bis 7 wird – abhängig davon, ob eine Funktion zur Behandlung eines Fehler-Ereignisses als letzter Parameter angegeben wurde – ein Objekt der Klasse `RAPRemoteObject` instanziiert. Diese Klasse ist eine Erweiterung der Flex-eigenen Klasse `RemoteObject` und enthält bereits einige Konfigurationsinformationen zum Einsatz mit BlazeDS. Diese sind für das Verständnis des clientseitigen RAP-Patterns jedoch unbedeutend. In Zeile 8 wird via Reflection ein Objekt instanziiert, das die aufzurufende Methode repräsentiert. Zeilen 9 bis 14 serialisieren das Parameter-Objekt in ein Array. Die Notwendigkeit für diesen Schritt ist dadurch gegeben, dass als Parameter eines entfernten Methodenaufrufs nur ein eingeschränkter Kanon an Datentypen möglich ist (siehe [Ado08]), nicht jedoch die Verwendung eigener Objekte. Im weiteren Verlauf des Codes (Zeile 15) erfolgt dann der Aufruf der gewünschten Methode[10].

5.3.2 Zusammenfassung und Bewertung des RAP-Patterns (client- und serverseitig)

Das clientseitige RAP-Pattern bringt zwei gewichtige Vorteile, denn:

1. Es führt den Gedanken der Zugriffsabstraktion fort, der bereits auf der Serverseite mit dem RAP-Pattern zum Tragen kommt (vgl. Abschnitt 5.2.2). Dadurch ergänzen sich Server und Client nicht nur in der Funktion, sondern auch in der Form.
2. Es ermöglicht eine einfache Austauschbarkeit des tatsächlich verwendeten Zugriffsmechanismus', denn zu diesem Zweck muss lediglich die konkrete Implementierung des Interfaces `IServiceConnector` ausgetauscht werden. Besonders einfach gestaltet sich ein solcher Austausch bei Verwendung des Programmierparadigmas *Dependency Injection*.

Auch serverseitig stellt das RAP-Pattern eine vielfältige Ergänzung zum jLAF-Framework dar. Es bietet eine sehr bequem zu verwendende Möglichkeit, Inhalte zur Verwendung in RIA-Clients aufzubereiten. Die Stärken dieser Erweiterung liegen insbesondere

[10] Dass dieser Methodenaufruf nur als Kommentar im Programmcode enthalten ist, hat seine Ursache darin, dass für diesen Aufruf zunächst noch eine gesonderte Handhabung der Parameterliste erfolgen muss. Diese trägt inhaltlich aber nichts Substanzielles zu den Ausführungen bei, sodass sie aus Gründen der Übersichtlichkeit, und um nicht vom Wesentlichen abzulenken, nicht gezeigt wird.

- in der Unabhängigkeit von verwendeten Kommunikationskanälen (z.B. via Servlet-Kommunikation oder via Remoting) sowie
- in der Möglichkeit, Inhalte für unterschiedliche Formate aufzubereiten (z.B. in HTML, XML, JSON oder ggf. auch als DAOs beim Einsatz von Remoting).

Vor allem aber wird durch diese Erweiterung das eingangs definierte Ziel erreicht, Inhalte unabhängig von ihrer späteren Repräsentation vorzuhalten und somit insbesondere HTML-Fallback-Lösungen aus Inhalten statischer RIAs automatisch generieren lassen zu können.

Die Erklärungen erfolgten in diesem Kapitel bis jetzt in einem abstrakten und somit allgemeingültigen Kontext. Somit ließen sich die zugrunde liegenden Ideen und deren Wiederverwendbarkeit in unterschiedlichen Umgebungen zwar herausstellen, jedoch nicht veranschaulichen. Dazu – und um die Praxistauglichkeit des RAP-Patterns unter Beweis zu stellen – sei auf das am Ende dieses Kapitels noch folgende Praxisbeispiel verwiesen. Dieses umfasst dann auch die Anwendung des Konzeptes zur Unterstützung von Mehrsprachigkeit, welches im folgenden Unterabschnitt vorgestellt wird.

5.3.3 Unterstützung von Mehrsprachigkeit: Das MLL-Pattern und dessen Integration im jLAF

Schon mehrfach wurde in dieser Arbeit auf die Bedeutung der Unterstützung mehrerer Sprachen hingewiesen. Sowohl im Hinblick auf Barrierefreiheit ("Sprachbarrieren") als auch als Merkmal für die Individualisierbarkeit einer Applikation ist das Vorhalten von *lokalisierten*, d.h. in Sprache und sonstiger Erscheinung[11] angepassten Versionen, von Bedeutung. Zur Begriffsverwendung sei an dieser Stelle bemerkt, dass der Terminus *Lokalisierung* von dem Terminus *Internationalisierung* unterschieden werden muss. Denn nach [Ber04] meint letzterer die "Vorbereitung einer Applikation, indem alles, was in verschiedenen geografischen Regionen unterschiedlich sein wird, identifiziert wird, und Maßnahmen getroffen werden, um anstelle von im Quelltext verankerten Werten für all dies unterschiedliche Versionen bereitzustellen" (eigene, sinngemäße Übersetzung). Im Unterschied dazu bedeutet nach [Ber04] *Lokalisierung* hingegen den Prozess des Anpassens der bei der Internationalisierung identifizierten Stellen an regionale Gepflogenheiten. Um die mitunter schmale Gratwanderung zwischen diesen beiden Begriffen zu vermeiden und weil es die tatsächliche Intention dieses Abschnitts präziser trifft, wird im folgenden der Begriff der Unterstützung von Mehrsprachigkeit verwendet.

Wie das jLAF-Framework im HTML-Kontext Mehrsprachigkeit unterstützt, wurde in Abschnitt 3.5.7 (Seite 75 ff.) dargestellt. Im aktuellen Abschnitt steht jedoch die Betrachtung der Client-Seite im Vordergrund, diese profitiert jedoch nicht unmittelbar von der Mehrsprachigkeit des (serverseitigen) Frameworks. Deshalb wurde im Rahmen dieser Arbeit auch eine clientseitige Unterstützung von Mehrsprachigkeit konzipiert. Das Resultat liegt in Form des "MLL-Patterns" (wobei MLL für *Multi Language Labeler* steht) vor.

Bei der Konzeption des MLL-Patterns stand die Erfüllung der nachfolgenden Anforderungen im Vordergrund:

- Die Lösung ist als clientseitige Ergänzung des (serverseitigen) jLAF-Framework gedacht.

[11] Unter dem Begriff "sonstige Erscheinung" kann beispielsweise die Schreibweise von Datumsangaben oder Nachkommazahlen verstanden werden. In manchen Sprachen ist die Verwendung eines Kommas als Trennzeichen der Nachkommastellen sowie des Punktes als Tausender-Trennzeichen üblich, in anderen verhält es sich genau umgekehrt.

- Sie soll unabhängig von einem speziellen RIA-Framework und einer Programmiersprache zu verwenden sein und somit flexibel für alle RIA-Frameworks eingesetzt werden können.
- Es steht nicht nur ein simples Übersetzen von Beschriftungen der Steuerelemente der RIA im Vordergrund. Vielmehr sollen auch die darzustellenden Inhalte jeweils in der gewünschten Sprache bezogen werden. Dies kann insbesondere auch bedeuten, dass je nach eingestellter Sprachversion andere Bilder (z.B. Hintergrundbilder) verwendet werden sollen.
- Es wird Wert gelegt auf eine einfache Erweiterbarkeit der zur Verfügung stehenden Sprachen: sie dürfen nicht fest im Programmtext verankert sein, sondern sollen eine Erweiterung (ggf. auch Austauschbarkeit) im laufenden Betrieb erlauben.
- Die verwendete Sprache soll in der RIA zur Laufzeit beliebig oft ein- und umgestellt werden können (kein Neuladen der RIA!).
- Der Bezug der Sprachinformationen soll im Programmcode abstrakt formuliert sein, sodass diese Informationen aus Dateien, ebenso gut aber auch aus Datenbanken oder anderweitigen Service-Providern bezogen werden können.
- Das Konzept soll so weit wie möglich gekapselt werden und dadurch eine saubere und übersichtliche Programmierung erlauben. Insbesondere soll im Praxisbeispiel (das in Flex entwickelt wurde) ein "Chaos" durch die Berücksichtigung unterschiedlicher Sprachen in den MXML-Dateien vermieden werden.

5.3.3.1 Native Unterstützung von Mehrsprachigkeit und Lokalisierungen in RIA-Frameworks

Die RIA-Frameworks bringen jeweils eigene Mechanismen mit, welche zur Unterstützung von Mehrsprachigkeit und Internationalisierung verwendet werden können. Auch das für die praktischen Implementierungen in dieser Arbeit verwendete Flex-Framework unterstützt die Internationalisierung. Während der Entstehung dieser Arbeit wurde das Flex-Framework in Version 3 veröffentlicht. Diese Version bringt im Vergleich mit der Vorgänger-Version enorme Verbesserungen im Umgang mit Internationalisierungsaufgaben mit. Oder anders formuliert: Version 2, welche zunächst für die Implementierungen des Praxis-Teils dieser Arbeit zur Verfügung stand, bot nur rudimentäre Unterstützung von Internationalisierung und Mehrsprachigkeit, sodass eine eigene Unterstützung dieser Funktionalität erarbeitet wurde. Mit dem Upgrade auf Version 3 ist die "hauseigene" Mehrsprachigkeit zwar deutlich verbessert worden (siehe dazu insbesondere [Smi07] sowie das Kapitel über Mehrsprachigkeit in Flex in [Mü09]), jedoch erweist sie sich als zu stark einengend für die mit dieser Arbeit verfolgten Ziele.

Von den in Flex nativ verwendeten Ansätzen unterscheiden sich auch diejenigen in Silverlight und Java FX nicht grundlegend. Stets erwarten sie, dass pro Sprache eine Properties-Datei vorliegt, in welcher in Form einer Schlüssel-Werte-Notation die Bezeichner der GUI-Elemente und deren jeweilige Beschriftung in der gewünschten Sprache enthalten sind. Diese Lösung ist insoweit einschränkend, als sie – mit Ausnahme des drittletzten Punktes der auf Seite 122 genannten Liste – die gewünschten Anforderungen nicht erfüllen können, sodass die Entwicklung einer eigene Lösung geboten erschien. Sie wird nachstehend vorgestellt.

5.3.3.2 Das MLL-Pattern

Das MLL-Pattern sieht drei Akteure vor:

1. ein Objekt der Klasse `MultiLanguageLabeler` (im Folgenden mit MLL abgekürzt), welches intern die verschiedenen Sprachen und Lokalisierungs-Informationen verwaltet und nach außen einen gekapselten Zugriff darauf anbietet,

2. einen GUI-Container (also ein Element einer GUI, dessen Aufgabe in der Verwaltung anderer GUI-Elemente liegt), dessen Inhalte auf Änderungen der Spracheinstellungen reagieren sollen[12] sowie

3. eine Datei[13] mit den Übersetzungen in alle gewünschten Sprachen.

Die wesentliche Idee hinter dem MLL-Pattern ist, dass der GUI-Container (genau) ein Objekt der Klasse `MLL` als private Membervariable besitzt. Sofern die durch den Container verwalteten GUI-Elemente sprachsensitiv sind[14], beziehen sie Informationen zu ihrer Darstellung indirekt, nämlich über das MLL-Objekt. Bei der Initialisierung des GUI-Containers bekommt er die Information übergeben, aus welcher Quelle (in der Regel wird dies eine Datei sein) die zu verwaltenden Sprachinformationen zu beziehen sind. Die angegebene Quelle wird an den Konstruktor des MLL-Objekts weitergegeben, welcher daraufhin das MLL-Objekt instanziiert und mit den aus der angegebenen Quelle gewonnenen Informationen ausstattet. Ist dies erfolgreich geschehen, so erhalten alle verwalteten GUI-Elemente des Containers die darzustellenden Inhalte durch das MLL-Objekt.

Der Aufbau einer Sprach-Definitionsdatei ist vergleichbar mit demjenigen des nativen Flex-Mechanismus' zur Unterstützung von Mehrsprachigkeit. Derartige Definitionsdateien enthalten Schlüssel-Werte-Paare, wobei der Schlüssel der im Quellcode verwendete Bezeichner und der zugehörige Wert der gewünschte Text in einer bestimmten Sprache ist. Listing 5.7 zeigt eine solche Datei, wobei hier ein erster Unterschied zur nativen Flex-Lösung auffällt: die Datei enthält Definitionen für *alle* Sprachen. Anstelle der sonst üblichen Aufteilung im Sinne von "pro Sprache eine Datei" können im MLL-Pattern alle sprachrelevanten Informationen an einer zentralen Stelle verwaltet werden (beispielsweise in dieser Datei). Wie im weiteren Verlauf dieser Arbeit noch gezeigt wird, muss diese zentrale Stelle aber keinesfalls eine Datei sein, sondern kann auch in Form einer Datenbank oder eines (Web-)Services vorliegen.

Listing 5.7: Beispiel für eine Sprachdefinitionsdatei (englisch/deutsch)

```
1  :language:EN
2  :btnSend:Send text
3  :btnCancel:Cancel
4  :lblPrompt:Please enter your message.
5  :lblConfirmation:Your message has been sent successfully.
6  :txtContent:@getPlaintext(texts/content.txt)@
7  :language:DE
8  :btnSend:Text senden
9  :btnCancel:Abbrechen
10 :lblPrompt:Bitte geben Sie Ihre Nachricht ein
11 :lblConfirmation:Ihre Nachricht wurde erfolgreich versandt
12 :txtContent:@getPlaintext(texts/contentDE.txt)@
```

[12] Aus Gründen der sprachlichen Einfacheit wird hier der Singular verwendet: *ein* GUI-Container. Selbstverständlich können auch mehrere GUI-Container zugleich mit dem MLL-Pattern verwendet und sprachlich gesteuert werden. In diesem Fall gelten die Erklärungen für jeden Container einzeln.

[13] Später wird noch gezeigt, dass anstelle einer Datei auch andere Möglichkeiten zur Persistierung der Informationen verwendet werden können (vgl. Abschnitt 6.5 auf Seite 177ff.).

[14] Mit der Bezeichnung *sprachsensitiv* sei hier gemeint, ob sich ein GUI-Element an Spracheinstellungen anpasst.

Ferner fällt in Listing 5.7 auf, dass hier abermals das bereits in Abschnitt 3.5.5 auf Seite 72ff. vorgestellte Direktivenkonzept zum Tragen kommt: wird zur Laufzeit als Wert eine Direktive gefunden (wie im Beispiel in den Zeilen 6 und 12), so wird diese ausgeführt und der Rückgabewert der als Direktive verwendeten Methode als Beschriftung (Wert) verwendet. An dieser Stelle ist die Integration des MLL-Patterns in das (serverseitig) vorhandene jLAF besonders gut erkennbar: abhängig von der eingestellten Sprache können unterschiedliche RAPs angesprochen werden und somit ein sprachabhängiges Verhalten der RIA erreicht werden. Die Notation von Direktiven in einer Sprachdefinition ist demnach das RIA-Analogon zur Beeinflussung der sprachrelevanten HTML-Generierung mit Hilfe von Konfigurationsdateien, wie sie an früherer Stelle (siehe Abschnitt 3.5.7 auf Seite 75) erläutert wurde.

Aufgrund der Integration des Direktivenkonzepts in das MLL-Pattern kann der Wert der Schlüssel-Werte-Paare einer Sprachdefinitionsdatei nicht nur die sprachabhängige Beschriftung enthalten, sondern auch eine auszuführende Aktion. Deshalb lässt sich nicht nur die textuelle Beschriftung von GUI-Elementen in der eingestellten Sprache erreichen, sondern vielmehr können auf diese Weise auch Inhalte aus anderen Quellen sprachabhängig bezogen werden, insbesondere HTML-Fragmente oder XML-Dokumente, aber auch Bilder bzw. grafische Beschriftungen. Zur Laufzeit der RIA werden dann alle sprachabhängig erhaltenen Inhalte zu einer Darstellung aggregiert. Eine derartige Flexibilität kann durch die native Unterstützung von Mehrsprachigkeit des Flex-Framworks nicht erreicht werden, sodass – allen Verbesserungen dieser Unterstützung in der neueren Flex-Version zum Trotz – das MLL-Pattern diesen überlegen ist. Erst recht gilt dies, wenn dem MLL-Pattern auf der Client-Seite das jLAF-Framework auf der Serverseite gegenübersteht.

Das konkrete Vorgehen zur praktischen Verwendung des MLL-Patterns soll nun anhand eines Beispiels erklärt werden. Listing 5.8 zeigt einen GUI-Container (hier: ein `Application`-Objekt, das zugleich auch die gesamte Anwendung repräsentiert), der eine private Instanz eines MLL-Objektes besitzt (Zeile 12). Zeile 14 nennt die Quelle der Sprachinformationen (hier: in Form eines Dateinamens), und in Zeile 18 wird – beim Initialisieren des GUI-Containers – das MLL-Objekt instanziiert. Der Konstruktor bekommt dabei zwei Parameter übergeben: einerseits die Quelle der Sprachinformationen sowie andererseits die Default-Sprache, die immer dann verwendet wird, wenn (noch) keine Informationen zu oder über eine(r) gewünschte Sprache vorliegen.

Listing 5.8: Verwendung eines MLL-Objekts im GUI-Container

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
       width="353" height="238" initialize="onInit()">
3    <mx:TextArea x="59.5" y="53" width="236" height="125" id="txtMessage"/>
4    <mx:Button x="59" y="196" label="{mll.getComponentLabel(btnSend)}" id="btnSend
       " click="onSend()"/>
5    <mx:Button x="230.5" y="196" label="{mll.getComponentLabel(btnCancel)}" id="
       btnCancel" click="txtMessage.text=''" />
6    <mx:Label x="59" y="27" text="{mll.getComponentLabel(lblPrompt)}" width="194"
       id="lblPrompt"/>
7    <mx:Script>
8      <![CDATA[
9        import mx.controls.Alert;
10
11       [Bindable]
12       private var mll: MultiLanguageLabeler;
13
14       private const labelFile: String = "demo.lbl"; // language definition file
15
16       private function onInit() : void {
```

```
17          addEventListener("GotLabels", onGotLabels);
18          mll = new MultiLanguageLabeler(labelFile, "DE");
19        }
20
21        private function onGotLabels(ev: Event) : void {
22          initialize();
23        }
24
25        private function onLanguageChange(newLanguage: String) : void {
26          mll.setLanguage(newLanguage);
27        }
28
29        private function onSend() : void {
30          // call method to send message...
31          Alert.show(mll.getLabelByName("lblConfirmation"));
32        }
33      ]]>
34    </mx:Script>
35  </mx:Application>
```

Das MLL-Pattern ist event-basiert, d.h. die über MLL-Objekte verfügenden GUI-Klassen reagieren auf das Eintreten "bestimmter" Ereignisse. Das wesentliche Ereignis in diesem Zusammenhang ist das Umstellen der aktuell verwendeten Sprache auf eine andere Sprache (`LanguageChangeEvent`), ein weiteres bedeutsames Ereignis ist der Erhalt der Sprachinformationen aus der genannten Quelle (`GotLabelsEvent`). Der Bezug dieser Informationen aus der genannten Quelle erfolgt in Form eines asynchronen Aufrufs, sodass die Applikation nicht auf die einzelnen (und je nach Umfang der Applikation zahlreich vorhandenen) Lesezugriffe warten muss, sondern eine erfolgreiche Durchführung der Leseoperationen durch entsprechende Ereignisse signalisiert bekommt. Die Ereignisbehandlung für den Erhalt von Sprachinformationen zeigt die Methode `onGotLabels()` (Zeilen 21 bis 23). Sie kann hier extrem einfach gehalten werden und besteht lediglich aus dem Aufruf der Methode `initialize()` des GUI-Containers, welcher zur Folge hat, dass er seine verwalteten GUI-Elemente aktualisiert. An dieser Stelle setzt sich das Prinzip der Ereignisbasierung des MLL-Patterns fort, denn die Eigenschaften dieser GUI-Elemente, welche letztlich den darzustellenden Inhalt repräsentieren, sind an das MLL-Objekt gebunden. Dies kommt durch die Notation mit geschweiften Klammern zum Ausdruck (exemplarisch hier in den Zeilen 4 und 6 bei der Eigenschaft `label` verwendet und durch das Schlüsselwort `Bindable` bei der Deklaration des MLL-Objekts in Zeile 11 eingeleitet). Die Ereignisbehandlungsroutine `onLanguageChange()` (Zeilen 25 bis 27) setzt die neue Sprache im MLL-Objekt durch den Aufruf der entsprechenden öffentlichen Methode. Das MLL-Objekt bezieht dann die Sprachinformationen in der neuen Sprache und löst das Ereignis `GotLabelsEvent` aus, was – wie beschrieben – dazu führt, dass sich die GUI-Elemente aktualisieren.

Listing 5.9: Öffentliche Methoden der Klasse `MultiLanguageLabeler`

```
1 public class MultiLanguageLabeler extends EventDispatcher
2 {
3   public function MultiLanguageLabeler(labelFile: String, curLang: String) {}
4   public function setLanguage(newLang: String) : void {}
5   public function getComponentLabel(ui: UIComponent) : String {}
6   public function getLabelByName(labelCand: String) : String {}
7 }
```

Zum Abschluss der Erklärungen zum MLL-Pattern sollen mit Listing 5.9 die öffentlichen Methoden eines MultiLanguageLabeler-Objektes gezeigt werden. Sie haben die folgenden Aufgaben:

Konstruktor (Zeile 3): Ihm werden als Parameter der Dateiname der Label-Datei[15] sowie ferner die Default-Sprache übergeben.

Methode `setLanguage`: Der Aufruf dieser Methode erfolgt mit derjenigen Sprache als Parameter, in der ab sofort der das MLL-Objekt besitzende GUI-Container (bzw. dessen GUI-Elemente) dargestellt werden soll(en). Wie zuvor beschrieben, werden die Label in der gewünschten Sprache intern "geholt" sowie nach erfolgreichem Abschluss dieser Aktion das Ereignis `GotLabelsEvent` ausgelöst.

Methode `getComponentLabel`: Die Methode durchsucht die Sprachdefinitionsdatei nach der passenden Beschriftung für das als Parameter übergebene GUI-Element und liefert diese Beschriftung zurück.

Methode `getLabelByName`: Diese Methode funktioniert im Wesentlichen analog zur Methode `getComponentLabel`, jedoch erhält sie als Parameter nicht das GUI-Element, zu welchem die Beschriftung geliefert werden soll, sondern stattdessen den (eindeutigen) Bezeichner eines solchen Elements.

Wie die Notation in Zeile 1 anzeigt, erbt die Klasse `MultiLanguageLabeler` von der Klasse `EventDispatcher`, damit sie über die Möglichkeit verfügt, Ereignisse auszulösen.

Die unterschiedlichen Anwendungsfälle der beiden zuletzt genannten Methoden können bereits in Listing 5.8 auf Seite 125 beobachtet werden: wenn – wie zuvor bereits beschrieben – die Beschriftungseigenschaften der GUI-Elemente mit Hilfe des *Data Bindings* im MXML-Teil gesetzt werden (Zeilen 3 bis 6), erfolgt dies mit Hilfe der Methode `getComponentLabel`. Die Methode `getLabelByName` kommt hingegen immer dann zum Tragen, wenn Beschriftungen von Elementen benötigt werden, die *nicht* im MXML-Teil definiert werden. Im Listing 5.8 ist dies in Zeile 31 der Fall, denn dort wird diese Methode benutzt, um eine Statusmeldung in einem PopUp-Fenster zu zeigen. Da dieses Fenster anonym erst zur Laufzeit erstellt wird, kann in Zeile 31 nicht mit einem Bezeichner darauf Bezug genommen werden, sodass anstelle der Methode `getComponentLabel` die Methode `getLabelByName` Verwendung findet.

Die an dieser Stelle noch bewusst abstrakt (und damit allgemeingültig) gehaltene Erklärung der Funktionsweise des MLL-Patterns wird am Ende dieses Kapitels im Rahmen der Vorstellung des Praxisbeispiels anhand eines konkreten Einsatzszenarios veranschaulicht (siehe Abschnitt 5.5 auf Seite 135).

5.3.4 Visuelle Individualisierbarkeit

Genau wie klassische HTML-Webseiten erlauben auch RIA-Frameworks die Verwendung externer StyleSheet-Dateien, um Design-Informationen von Inhalten zu trennen. Die zu diesem Zweck verwendeten Stylesheet-Dateien sind jedoch (z.B. bei Flex und Silverlight) proprietäre Erweiterungen des W3C-CSS-Standards und insoweit also nicht außerhalb des jeweiligen RIA-Frameworks zu verwenden. Dessen ungeachtet erlauben sie eine durchaus komfortable und flexible Konfiguration des verwendeten Designs und dessen Änderung zur Laufzeit.

Bewusst wird in diesem Abschnitt der Begriff "Design" anstelle von "Layout" verwendet, denn bei RIAs sind StyleSheet-Dateien primär dazu gedacht, die optische Ausgestaltung der GUI-Elemente zu definieren, nicht jedoch deren Positionierung innerhalb der RIA (die an

[15] Wie bereits mehrfach bemerkt, wird an späterer Stelle in dieser Arbeit statt des Lesens aus einer Datei auch das Holen der Sprach-Informationen aus einer internen Datenbank möglich sein.

dieser Stelle durchaus als GUI gesehen werden muss). In diesem Zusammenhang hat sich auch der Anglizismus "*Skinning*" etabliert. Damit unterscheiden sich StyleSheets in RIAs jedoch von traditionellen CSS-Dateien für HTML-Dokumente, denn diese definieren zugleich das *Layout* der Seite(n), also insbesondere auch die Aufteilung und Positionierung der einzelnen Bestandteile einer Seite.

Die Unterstützung von Barrierefreiheit wird durch die Anwendung der Möglichkeiten, die zum *Skinning* von RIAs bereitstehen, nur unwesentlich verbessert. Es ließe sich auf diese Weise zum Beispiel ein Layout verwenden, das mit hohen Kontrasten und/oder besonders großen Schriften operiert und somit sehschwachen Benutzern eine Hilfe darstellt. Betrachtet man statt der Barrierefreiheit die *Individualisierbarkeit* von RIAs, so trägt die Verwendung von StyleSheets bei RIAs durchaus zu dieser bei. In einem späteren Kapitel dieser Arbeit wird der Begriff der *Individualisierbarkeit* (einer RIA) in den Mittelpunkt der Betrachtungen gestellt (vgl. Abschnitt 6.1.2 auf Seite 154), sodass für eine präzise Einordnung dieses Begriffs auf das entsprechende Kapitel verwiesen sei.

StyleSheets lassen sich (in Flex) zur Laufzeit beliebig austauschen[16]. Dies ist durch nur eine Zeile Code möglich:

Listing 5.10: Austausch eines StyleSheets in Flex zur Laufzeit

```
1 StyleManager.loadStyleDeclarations("myStyle.css", true);
```

Auf diese Weise kann zu einer RIA eine Vielzahl verschiedener Layouts angeboten werden, aus denen der Benutzer dann zur Laufzeit dasjenige auswählt, das seinen Wünschen und Anforderungen am stärksten gerecht wird. Dies für sich genommen ist keine neue Technik, und dies wurde im Rahmen dieser Arbeit auch nicht weiterentwickelt. Aus Gründen der Vollständigkeit – und weil es ein "Puzzle-Teil" des in Kapitel 6 eingeführten Konzepts der Individualisierbarkeit von RIAs darstellt – soll es hier jedoch kurz erwähnt werden.

5.4 Fallback-Lösungen im Umfeld traditioneller und JavaScript-basierter Webseiten

In diesem Kapitel stehen RIAs der Kategorie 2 (gemäß der Definition auf Seite 83) im Mittelpunkt der Betrachtungen. Insbesondere sind die aufgezeigten Lösungsmöglichkeiten für diese Kategorie konzipiert. Der Vollständigkeit halber soll nun noch kurz darauf eingegangen werden, welche Fallback-Möglichkeiten bei auf JavaScript basierenden Webseiten (welche ja für RIAs der Kategorie 1 eine bedeutende Rolle spielen) zur Verfügung stehen und welchen historischen Hintergrund diese Lösungen haben.

Der historische Hintergrund der Konzepte ist insoweit interessant, als sie im Kern stets auf die gleichen Problemstellungen führen, die auch heutzutage noch aktuell sind (wenn auch unter technologisch anderen Rahmenbedingungen). Es geht stets um die Frage, wie Seiten dann noch zuverlässig dargestellt werden können, wenn die Ausgabegeräte einen Teil der von den Seiten vorausgesetzten Technologien nicht unterstützen.

Ein naiver Ansatz war es vor einigen Jahren, die Bereitstellung zweier Versionen von einer Seite zu fordern: einmal die "normale" Version und einmal eine rein textuelle. In seinem Blog diskutierte Mike Davies bereits 2004 das Für und Wider derartiger *Text Only*-Versionen von Webseiten und kam dabei zu folgendem Schluss:

[16] Dazu müssen sie zuvor in eine SWF-Datei compiliert worden sein, d.h. sie liegen zur Laufzeit nicht mehr als rein textuelle CSS-Datei(en) vor.

> *"A text-only version of the website effectively segregates the audience it is targetted at. History demonstrates that segregation does not lead to equality, but to stigmatisation. South Africa under the regime of Apartheid is a clear example of this stigmatisation."*[Dav04b]

Ferner führte er aus, dass das Vorhalten zweier oder mehrerer Versionen auch zu einem Problem in Zusammenhang mit Suchmaschinen führen würde (Welche Version soll gefunden werden? Welche wird *de facto* gefunden?), und stellte fest, dass Text-Only-Varianten in der Regel nur über die vollwertigen Seiten erreicht werden können, sodass sie hinter diesen gewissermaßen "versteckt" sind. Davies erkennt somit ein Paradoxon: Text-Only-Seiten können also nur erreicht werden, wenn der Benutzer sie über die Seiten aufruft, die sie eigentlich deswegen ersetzen sollen, weil sie der Benutzer *nicht* verwenden kann.

Argumente wie diese sowie noch viele weitere führten zu einem anderen Denken in Bezug auf den Umgang mit barrierefreien Fallback-Lösungen. Von nun an stand die Frage im Mittelpunkt, wie eine Seite dergestalt konzipiert werden kann, dass sie nur in einer einzigen Version vorliegt, aber dennoch eine gleichermaßen gute Verwendbarkeit bietet, ganz gleich mit welchem Ausgabegerät man sie aufruft. In diesem Zusammenhang sind zwei Paradigmen entstanden: *Graceful Degradation* (GD) und *Progressive Enhancements* (PE).

5.4.1 Graceful Degradation und Progressive Enhancement

Die in diesem Kapitel betrachtete Notwendigkeit, Fallback-Versionen von Webseiten vorzuhalten, ist insoweit also keine aktuelle Problemstellung. Das Vorhalten spezieller Seiten-Varianten für den Fall, dass einige Ausgabegeräte Inhalte nicht fehlerfrei oder mitunter auch gar nicht darstellen können, kann bereits seit vielen Jahren als geübte Praxis angesehen werden. Wesentlich zu der Verbreitung dieses Gedankens hat sicherlich der sogenannte "Browserkrieg" beigetragen, während dessen zwei Hersteller von Internet-Browsern um die Marktvorherrschaft kämpften. Zwecks Abgrenzung gegeneinander statteten sie ihre Produkte mitunter mit besonderen Fertigkeiten (*Features*) aus, mit denen der jeweils andere Browser nicht aufwarten konnte. In der Konsequenz führte dies jedoch dazu, dass Webseiten, die von derartigen Funktionen Gebrauch machen wollten, nicht oder zumindest nicht im gleichen Umfang auf beiden Browsern darstellbar waren. Je nach Benutzerkreis wurde mitunter bewusst in Kauf genommen, dass die Seite nur in einem bestimmten Browser verwendbar sein wird.

Für Seiten, die im Internet der Allgemeinheit zur Verfügung stehen sollen, ist heutzutage die einseitige Fokussierung auf einen Browser nicht (mehr) zumutbar. Deshalb war und ist auch heute noch oftmals eine Fallunterscheidung des verwendeten Browsers nötig, die dann – abhängig vom Browsertyp – bestimmte Direktiven im HTML-Quellcode anspricht. Da diese Fallunterscheidungen aber ihrerseits wieder nicht mit HTML-Mitteln allein, sondern nur unter Verwendung von JavaScript möglich sind, erkauft man sich eine solche "Lösung" mit neuen Problemen (Einsatz von JavaScript). Ein vielversprechenderer Ansatz war und ist es deshalb, Fallback-Lösungen für die nicht vollständig unterstützten Browser bereitzustellen.

In Zusammenhang mit Fallback-Lösungen haben sich die zwei unterschiedlichen Design-Strategien *Graceful Degradation* und *Progressive Enhancement* etabliert, die auch heute noch von Bedeutung sind. Denn während damals die Unterstützung des einen oder des anderen Browsers der Gegenstand der Betrachtungen war, kann nun die Unterscheidung zwischen der "reinen" HTML-Version und ihrem reichhaltigeren AJAX- bzw. JavaScript-Pendant vorgenommen werden. Beiden Strategien gemein ist, dass sie jeweils Empfehlungen geben, wie die Entwicklung

eines Webauftritts im Hinblick auf Kompatibilität mit älteren Browsern sowie alternativen Ausgabegeräten erfolgen soll. Sie gehen beide davon aus, dass es verschiedene Versionen eines Webauftritts gibt bzw. geben muss, damit das Spektrum von stark vereinfachter Darstellung (für ältere Browser und alternative Ausgabegeräte) bis hin zur technisch modernsten Variante adressiert werden kann. Der wesentliche Unterschied zwischen den beiden Strategien ist jedoch die Frage, ob man aus der technisch ausgereiftesten Version sukzessive die vereinfachte Darstellung gewinnt (GD-Ansatz) oder den umgekehrten Weg beschreitet und durch systematisches Hinzufügen die technisch ausgereifteste Lösung aus einer stark vereinfachten Version gewinnt (PE-Ansatz).

5.4.1.1 Zur aktuellen Bedeutung der dargestellten Prinzipien

Der erwähnte "Browserkrieg", aus dem die genannten Strategien hervorgegangen sind, ist zwar seit einigen Jahren vorbei, jedoch bedeutet dies längst nicht, dass damit die vorstehenden Betrachtungen obsolet geworden sind. Zwar hat der Marktdruck dazu geführt, dass alle gängigen Browser standardkonform geworden sind und sie deshalb nach diesen Standards entwickelte Seiten auch in gleicher Weise anzeigen. Andererseits sind aber neben klassischen Browsern heutzutage viele alternative Ausgabegeräte für Webseiten in Verwendung, wobei hier insbesondere wieder mobile Geräte von Bedeutung sind. High-End-Geräte dieses Genres beinhalten zwar bereits die Unterstützung aller gängigen Standards, ältere oder weniger ausgereifte Geräte hingegen sind mit komplexeren Webseiten überfordert. Deshalb müssen auch für sie Möglichkeiten bereitstehen, die betreffenden Seiten in einer vereinfachten Form dazustellen. Der Trend, immer mehr Geräte der Unterhaltungselektronik auch als Internet-Browser einzusetzen (insbesondere mobile Geräte mit dementsprechend kleinen Displays), lässt deshalb den GD- und PE-Ansätzen durchaus Aktualität zukommen.

5.4.1.2 Graceful Degradation

Dieser Ansatz geht davon aus, dass ein Webauftritt für einen bestimmten Browser bzw. einen bestimmten Kanon von zur Verfügung stehenden Technologien entwickelt und optimiert werden soll, wobei stets die leistungsfähigsten und modernsten Systeme adressiert werden. Ist der Webauftritt für derartige Zielsysteme fertiggestellt, wird in nachfolgenden Entwicklungsschritten geprüft, wie Zielsysteme mit weniger Fähigkeiten mit dem Webauftritt zurecht kommen. Es werden dabei nach und nach Anpassungen derart vorgenommen, dass auch weniger fähige Zielsysteme noch eine akzeptable Ausgabe zur Verfügung stellen. Die Anpassung erfolgt beim *Graceful Degradation* also vom besten zum immer schlechter werdenden Zielsystem, wobei eine andere Sichtweise anstelle von "das beste" bzw. "das schlechtere" Zielsystem auch "das am weitesten verbreitete" und "ein weniger verbreitetes" Zielsystem nennt (siehe dazu z.B. [Ols07]).

Mit zwei Standard-Literatur-Beispielen soll das Konzept der *Graceful Degradation* kurz erläutert werden:

Das `alt`-Attribut: Die korrekte Verwendung des HTML-Attributs `alt` in Zusammenhang mit Bildern (`img`-Tag) bewirkt, dass Zielgeräte, die keine Grafiken anzeigen können, stattdessen den im `alt`-Attribut genannten Alternativ-Text ausgeben (vgl. dazu auch Abschnitt 2.6.2 auf Seite 14). Browser bzw. Zielgeräte mit Grafikfähigkeiten werden als Regelfall angesehen und führen zur bestmöglichen Ausgabe, nämlich dem Anzeigen des Bildes bzw. der Grafik. Deshalb wird die Webseite zunächst nur im Hinblick auf

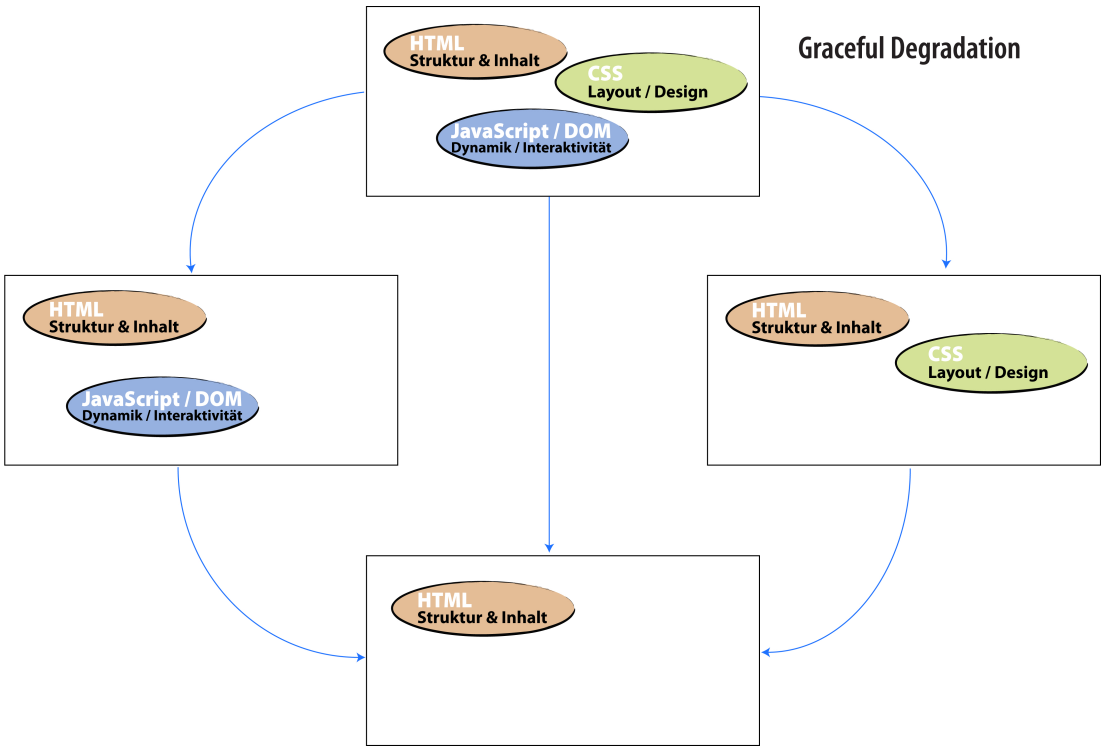


Abbildung 5.4: Rückfall auf einfachere Versionen bei *Graceful Degradation*

eine Darstellung auf grafikfähigen Ausgabegeräten konzipiert. Der Einsatz alternativer Ausgabemedien führt dann stattdessen zur Darstellung von (Alternativ-)Text. Inhaltlich kann durch einen geeignet gewählten Alternativtext die Aussage der Grafik ebenfalls wiedergegeben werden, jedoch ist diese Form im Vergleich zur grafischen Darstellung schlechter ("*degraded*").

Das `noscript`-Tag: HTML kennt das Sprachelement `noscript`, welches eine alternative Ausgabe für den Fall deklariert, dass das Zielgerät JavaScript nicht beherrscht. Auch hier stellt zunächst der Fall, dass JavaScript verfügbar ist, den optimalen Fall dar. Trifft dieser nicht zu, kann durch die Verwendung des `noscript`-Tags eine vereinfachte Alternative ausgegeben werden.

Dem Konzept der *Graceful Degradation* liegt die Annahme zugrunde, dass mit der Zeit die Zielsysteme (Browser) immer besser und leistungsfähiger würden und insofern ein Entwickeln für die aktuellsten und umfangreichsten Technologien zu bevorzugen ist. Mit dem Erstarken von sehr kleinen (mobilen) Geräten für den Internetzugang (PDAs, Mobiltelefone, etc.) erwies sich diese Annahme jedoch als falsch, denn die Leistungsfähigkeit kleiner und mobiler Geräte war (und ist) nicht mit den Möglichkeiten aktueller Browser vergleichbar.

5.4.1.3 Progressive Enhancement

Im Jahr 2003 schlug Steve Champeon mit seinem Konzept des *Progressive Enhancement (PE)* eine vollkommene Umkehrung des Graceful-Degradation-Prinzips vor (siehe [CF03]). Sein PE-Prinzip startet mit der Betrachtung der kleinsten Schnittmenge von Fähigkeiten der potenziellen Ausgabemedien (Champeon nennt dies den "*lowest common denominator of browser*

software functionality"). Schritt für Schritt werden dann Verbesserungen bzw. Erweiterungen für bessere Ausgabemedien hinzugefügt (siehe Abbildung 5.5). Kann ein Gerät diese Erweiterungen nicht verarbeiten, so ist dies kein gravierender Nachteil, da sichergestellt ist, dass die "unterste" Stufe dennoch eine ausreichende Darstellung garantiert.

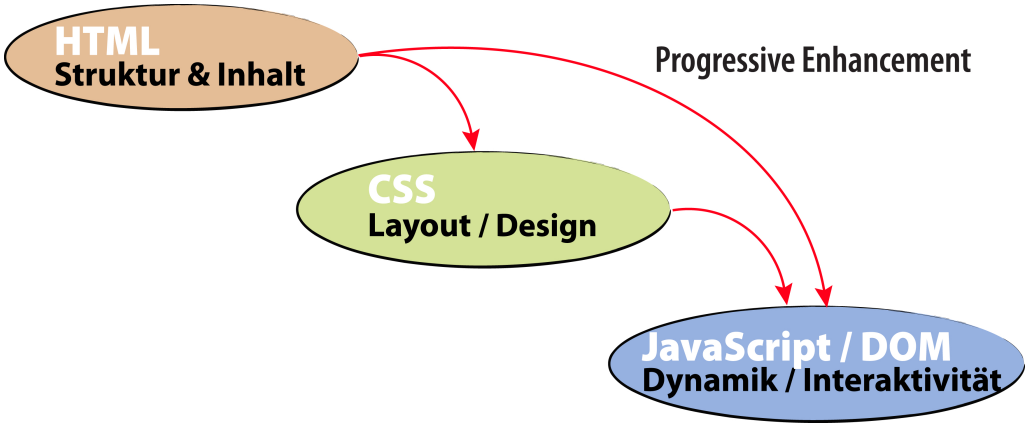


Abbildung 5.5: Sukzessive Erweiterungen bei *Progressive Enhancement*

Die in dieser Arbeit immer wieder erwähnte strikte Trennung von Layout und Inhalt ist bereits eine simple Form des *Progressive Enhancement*, denn wenn ein Ausgabemedium mit den StyleSheet-Informationen nicht umzugehen weiß oder braucht (zum Beispiel ein Braille-Display), so werden die Layout-Informationen einfach ignoriert, wohingegen die Struktur und der Inhalt der Seite ohne Einschränkung benutzt werden können.

5.4.2 Augmentative Authoring

Ungefähr zeitgleich mit Champeon hat Jukka Korpela sein Konzept des *Augmentative Authoring* vorgestellt (siehe [Kor02]), welches im Wesentlichen die gleiche Idee wie Champeons *Progressive Enhancement* verfolgt. Der Kerngedanke in Korpelas *Augmentative Authoring* ist es, eine Seite – bezogen auf je eine in ihr verwendete Technologiekomponente oder Funktionalität – stets für zwei Modi vorliegen zu haben: einmal für den Fall, dass die konkrete Komponente/Funktionalität aktiviert (d.h. verfügbar) ist und einmal dafür, dass sie deaktiviert (nicht verfügbar) ist. Korpela benennt diese Modi als "einfach" (*simple*) und "verbessert" (*enhanced*).

Auch beim *Augmentative Authoring* beginnt die Entwicklung einer Webseite mit Techniken, die von *jedem* Zielgerät beherrscht werden. Die Besonderheit dieses Konzepts – und damit auch die Abgrenzung zum *Progressive Enhancement* – liegt in der Methodik beim Umgang mit Erweiterungen. Korpela fordert hier die Beachtung zweier Prinzipien:

Dualitätsprinzip (*duality principle*): Jede Erweiterung einer Webseite sollte in zwei "Ausprägungen" durchgeführt werden: einfach (*simple*) und verbessert (*enhanced*). Die einfache Version kann hierbei bereits als Fallback-Lösung angesehen werden für den Fall, dass die verbesserte Variante nicht dargestellt werden kann oder soll.

Orthogonalitätsprinzip (*orthogonality principle*): Jede Erweiterung soll orthogonal zu allen anderen sein, d.h. sie soll aktiviert oder deaktiviert werden können, ohne eine andere zu erfordern oder zu beeinflussen.

5.4.3 Ein zusammenfassendes Beispiel für Graceful Degradation, Progressive Enhancement und Augmentative Authoring

Anhand eines HTML-Praxis-Beispiels sollen die vorgestellten grundlegenden Web-Engineering-Prinzipien noch einmal verdeutlicht werden. Es ist der Webseite von Dan Tobias[17] entnommen, auf welcher der Autor noch eine ausgiebige Diskussion dazu sowie weitere (ähnliche) Beispiele erläutert.

Das Beispiel zeigt eine populäre Verwendung von *PopUp*-Fenstern, jenen Einblendungen, die – in der Regel kontextsensitiv – Informationen einblenden. Einerseits können diese Informationen reine Werbung beinhalten und vom Benutzer unerwünscht sein. (Gerade diese Werbe-PopUp-Fenster haben JavaScript einen schlechten Ruf eingebracht und zum Abschalten der JavaScript-Fähigkeit im Browser geführt.) Andererseits lassen sich PopUp-Fenster durchaus auch sinnvoll einsetzen. Das nachstehende Beispiel möge einen solchen sinnvollen Einsatz motivieren: der Benutzer soll beim Anklicken eines Links auf der Webseite in einem sich dann öffnenden PopUp-Fenster ergänzende Erklärungen zu einem bestimmten Thema finden. Der Erklärungstext sei dabei in der Datei `explanations.html` abgelegt[18].

HTML bietet keine Techniken, um derartige PopUps zu realisieren, hierfür ist Verwendung von JavaScript nötig. Da aber nicht alle Ausgabegeräte mit JavaScript umgehen können (oder aus Sicherheitsgründen damit nicht umgehen sollen), muss im Interesse der Barrierefreiheit eine Vorkehrung getroffen werden, um die vom Benutzer durchaus erwünschten Informationen des PopUp-Fensters dennoch zur Darstellung zu bringen.

Einen naiven Ansatz, der auf funktionsfähige Fallback-Lösungen keine Rücksicht nimmt, stellt der Code in Listing 5.11 dar. Dabei sei `displayPopupFunction()` eine JavaScript-Funktion, welche das PopUp-Fenster erzeugt und die als Parameter übergebene HTML-Datei darin anzeigt.

Listing 5.11: naive (nicht empfehlenswerte) Fassung

```
1 <a href="javascript:displayPopupFunction('explanations.html')">
```

Das Problem des Codes aus Listing 5.11 besteht darin, dass Ausgabegeräte, die kein JavaScript unterstützen, den Link nicht zu behandeln wissen und ihn daher entweder gar nicht oder in einer nicht funktionsfähigen Art und Weise darstellen. In der Terminologie dieses Abschnitts heisst dies nichts anderes, als dass der dargestellte Code das Prinzip der *Graceful Degradation* nicht unterstützt.

Nähert man sich der Problemstellung aus der Sichtweise des *Progressive Enhancement*, so steht zunächst die Notation eines gewöhnlichen HTML-Links im Vordergrund (siehe Listing 5.12).

Listing 5.12: ein simpler HTML-Link

```
1 <a href="explanations.html">
```

Im zweiten Schritt wird der Link nun "verbessert" (*enhanced*) zu der in Listing 5.13 gezeigten Variante. Diese neue Version des Codes lässt *jedes* Ausgabegerät – unabhängig von seiner

[17] siehe `http://webtips.dan.info/graceful.html`

[18] Ein reales Szenario für den sinnvollen Einsatz von JavaScript-basierten PopUp-Fenstern ist z.B. das Ausfüllen eines Formulars. Das PopUp-Fenster kann dann zusätzlich zum eigentlichen Seiteninhalt (Formular) eingeblendet werden, um kontextsensitiv ergänzende Informationen wie z.B. Ausfüllhilfen bei der Navigation durch das Formular anzuzeigen. In diesem Fall würde die zu ladende Datei dynamisch in den HTML-Code generiert.

JavaScript-Fähigkeit – zunächst einen "normalen" Link erkennen (Zeile 1). Dieser Link verweist auf die Seite mit den Zusatzinformationen, die eigentlich im PopUp-Fenster angezeigt werden sollen. Kann das Ausgabegerät kein JavaScript, stellt es den Link zu dieser Seite dar und ermöglicht so, diese Seite aufzurufen. Die Fähigkeit, HTML-Code zu interpretieren, wird hier also als '*lowest common denominator of browser software functionality*" im Sinne des *Progressive Enhancement* angesehen. JavaScript-fähige Browser hingegen erkennen die Ereignisbehandlung `onClick` (Zeile 2) und rufen die dort notierte Funktion `displayPopupFunction(URL)` auf, sobald der Link angeklickt wird. Das nachgestellte `return false` (Zeile 3) dient dazu, dem eigentlichen Link *nicht* zu folgen, sondern nur die Ereignisbehandlung durchzuführen und damit das PopUp-Fenster einzublenden.

Listing 5.13: Progressive Enhancement eines Links mit JavaScript

```
1 <a href="explanations.html"
2     onClick="displayPopupFunction('explanations.html');
3     return false;">
```

Während Benutzer von Browsern, die JavaScript voll unterstützen, keinen Unterschied zwischen den Varianten aus Listing 5.11 und 5.13 bemerken werden, ist dies jedoch sehr wohl bei Benutzern von Ausgabegeräten *ohne* JavaScript-Fähigkeiten der Fall. Dort führt die Version aus Listing 5.11 dazu, dass der Link gar nicht angezeigt wird oder seine Form der Darstellung von der Interpretation des konkreten Ausgabegerätes abhängig ist, wohingegen die Variante aus Listing 5.13 auf eine Fallback-Lösung in reinem, standardkonformem HTML führt.

5.4.4 Unobstrusive JavaScript

Die rasante Verbreitung aktueller RIA-Frameworks der Kategorie 1 lässt der Frage nach dem korrekten Umgang mit JavaScript hohe Aktualität zukommen. Christian Heilmann hat dazu den Begriff *Unobstrusive JavaScript* geprägt (siehe [Hei05]), welcher in der Übersetzung eigentlich "unaufdringliches JavaScript" heissen müsste, von Heilmann aber mit "barrierefreies JavaScript" ins Deutsche übersetzt wurde. Mit diesem Begriff fasst Heilmann die Idee zusammen, Webseiten in drei Ebenen (*Layers*) aufzuteilen (die deutschen Begriffe sind hierbei eigene Übersetzungen):

1. Inhalts- bzw. Struktur-Ebene (*Structure Layer*)
2. Präsentations-Ebene (*Presentation Layer*)
3. Interaktions-Ebene (*Behaviour Layer*)

Der wesentliche Punkt in Heilmanns Ausführungen ist die strikte Trennung dieser drei Ebenen voneinander, insbesondere die Aufforderung, JavaScript nicht direkt in das HTML-Dokument zu schreiben ("*Never, under any circumstances add Javascript directly to the document.*"[Hei05]). Danach stellt beispielsweise Listing 5.13 auf Seite 134 bereits einen Verstoß gegen diese Trennung dar. Denn in diesem Codebeispiel ist die Struktur des Inhalts (d.h. der Link, Ebene 1) mit Informationen zur Behandlung bestimmter Ereignisse (hier: Mausklick, Ebene 3) vermischt. Um eine Trennung von Struktur und Interaktion zu erreichen, muss der Code – wie in Listing 5.14 gezeigt – modifiziert werden:

Listing 5.14: Beispiel für die Berücksichtigung des Paradigmas "Unobstrusive JavaScript"

```
1 <a href="explanations.html" class="popup">
```

In einer *externen* JavaScript-Datei wird dann ein Skript vorliegen, das allen Links der HTML-Datei, welche den (Klassen-)Namen `popup` haben, eine passende Ereignisbehandlungsroutine zuweist[19]. Wenn dieses Skript beim Laden der Seite aufgerufen wird, erhält man im Ergebnis wieder eine Seite, die – sofern JavaScript auch wirklich unterstützt wird – das gewünschte Verhalten zeigt. Andernfalls bleiben jegliche JavaScript-Anweisungen "unsichtbar", weil sie *nicht* in der semantischen Struktur des Dokuments vorliegen und auch – wegen der fehlenden JavaScript-Fähigkeit des aktuellen Ausgabemediums – nicht in das HTML-Dokument bei dessen Aufruf generiert werden (können). Sie bleiben in diesem Fall also unbemerkt, unberücksichtigt und damit *unobstrusive*: unaufdringlich.

5.5 Proof of Concept: Das Webbuch

Um die Praxistauglichkeit der in diesem Kapitel vorgestellten Konzepte unter Beweis zu stellen ("*Proof-of-Concept*") und ihre konkrete Verwendung zu veranschaulichen, schließt das Kapitel mit einem Praxisbeispiel, in dem sowohl die client- als auch die serverseitigen Konzepte Verwendung finden. Als (experimenteller) Betrachtungsgegenstand soll ein privater Webauftritt in Form einer persönlichen Homepage dienen, der sowohl als multimediale, statische RIA als auch als HTML-Fallback-Version vorgehalten wird. Die RIA-Variante der Homepage repräsentiert ein virtuelles Buch (im folgenden "Webbuch" genannt), das von den multimedialen Fähigkeiten von RIAs Gebrauch macht und eine besondere *User Experience* bieten möchte. Zugleich sollen die Inhalte des Webbuchs – unter Verwendung des jLAF-Framworks – automatisch in eine barrierefreie HTML-Fallback-Version des Webbuchs überführt werden. Abbildung 5.6 zeigt einen Screenshot der RIA-Version des Webbuchs[20].

5.5.1 Vorbemerkungen zum Webbuch

Das Webbuch wurde insbesondere deshalb als experimenteller Rahmen für diese Arbeit ausgewählt, weil es in vielerlei Hinsicht die wesentlichen Ideen und Absichten der in dieser Arbeit vorgestellten Konzepte aufgreift:

- Das Webbuch ist ein Repräsentant klassischer Web 2.0-Anforderungen, d.h. es wartet mit multimedialen Elementen (Texte, Fotos, Videos, Feedback- bzw. Blog- und Kommentarfunktion) auf.
- Es fällt in die Kategorie "statische RIA" (vgl. Definition auf Seite 106), denn es vermittelt Inhalte, die bereits fest vorliegen; lediglich die Handhabung des Webbuchs ist interaktiv ("*Interactive Design*").
- Als öffentlich verfügbare Homepage steht das Webbuch allen Internetnutzern zur Verfügung, d.h. die Notwendigkeit, die Darstellung an individuelle Bedürfnisse anpassen zu können, ist gegeben. Somit können die Konzepte zur Individualisierbarkeit gewinnbringend eingesetzt werden.
- Aus dem gleichen Grund besteht Interesse daran, die Inhalte des Webbuchs in verschiedenen Sprachen vorhalten zu können (aktuell in Deutsch und in Englisch), sodass die Unterstützung der Mehrsprachigkeit von Bedeutung ist.

[19] Die technische Realisierung erfolgt dabei durch Methoden der DOM-Manipulation.

[20] Die Online-Fassung des Webbuchs ist zu erreichen unter der URL `http://www.norbert-heidenbluth.de/webbook`.

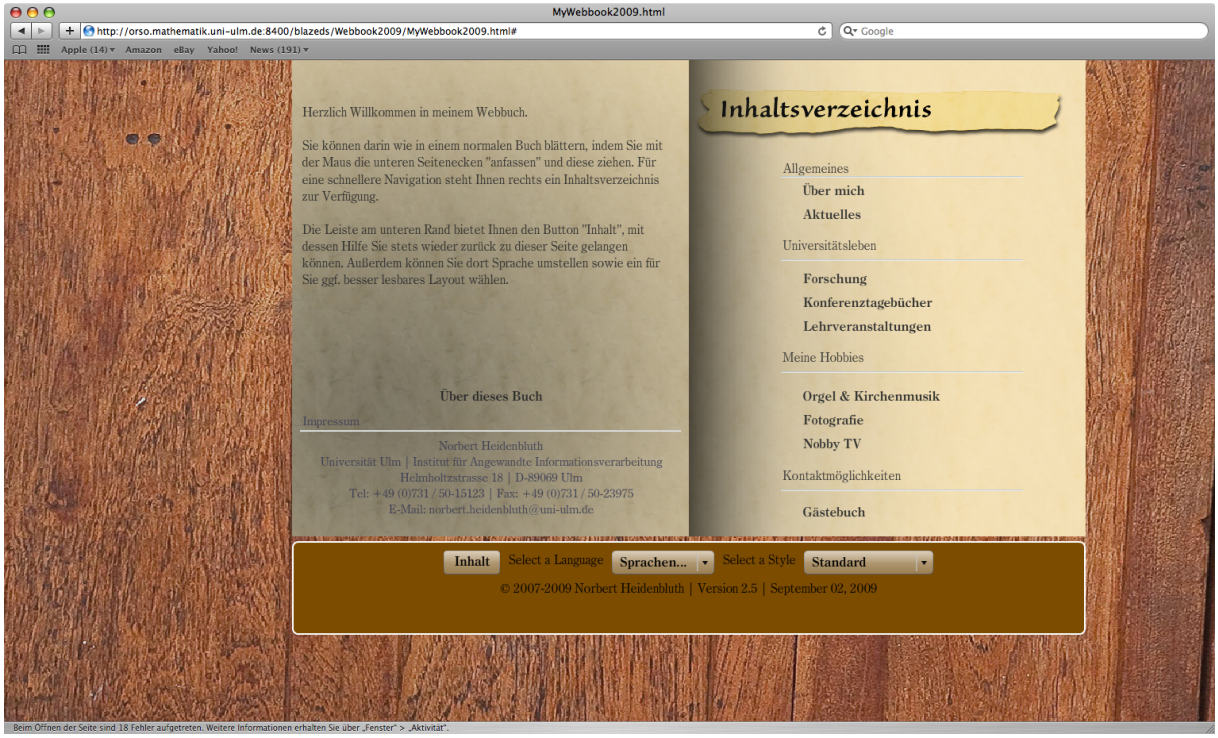


Abbildung 5.6: Screenshot der RIA-Version des Webbuchs (Inhaltsverzeichnis)

- Selbstverständlich soll der Webauftritt barrierefrei sein und auch von Suchmaschinen gefunden werden können. Aufgrund der Ausführungen im bisherigen Verlauf dieser Arbeit erwächst daraus der Bedarf an einer HTML-Fallback-Lösung des Webbuchs.

5.5.1.1 Zur Übertragbarkeit der Konzepte des Webbuchs

Das Webbuch – als prototypischer Repräsentant der auf Seite 106 definierten Kategorie der *statischen RIAs* – kann nicht nur als multimediale Variation einer persönlichen Homepage dienen, sondern beispielsweise auch als Online-Katalog eines Warenhauses verwendet werden. Insofern sei ausdrücklich bemerkt, dass die Konzepte zur Realisierung des Webbuchs nicht auf diesen speziellen Anwendungsfall beschränkt bleiben, sondern auf viele andere Einsatzszenarien, in denen RIAs der Kategorie 2 Verwendung finden sollen, übertragbar sind. Das beschriebene Praxisbeispiel ist hier also lediglich als *Proof-of-Concept* des jLAF-Frameworks zu sehen. Oder anders ausgedrückt: das jLAF-Framework erlaubt beliebige andere Implementierungen seiner Interfaces und abstrakten Klassen.

5.5.1.2 Zur Benennung der Klassen und Interfaces zwecks Unterscheidung des Frameworks vom Praxisbeispiel

Mit Hilfe der im Framework und im Praxisbeispiel verwendeten Bezeichnungen soll stets erkennbar sein, ob es sich um einen Bestandteil des Frameworks oder der konkreten (Beispiel-)Implementierung handelt: wenn Klassennamen die Bezeichnung `Webbook` enthalten, sind sie Bestandteil des Praxisbeispiels. Erläuterungen zu Klassen, die den Namensbestandteil `Webtool` führen, beziehen sich auf das Framework und *nicht* unmittelbar auf das Praxisbeispiel. Auf eine einfache Formel gebracht, bedeutet dies: das “Webbook” nutzt das “Webtool”.

Um einen Bezug zwischen den abstrakt gehaltenen Erklärungen in den Abschnitten 5.2 bzw. 5.3 und ihren Konkretisierungen im Praxisbeispiel deutlich hervortreten zu lassen, sind die nachfolgenden Erläuterungen erneut in server- und clientseitige Implementierungen unterteilt und folgen damit der Gliederung der genannten Abschnitte.

5.5.2 Serverseitige Implementierung

Der im Webbuch darzustellende Inhalt liegt vollständig getrennt von der RIA vor. Weil es sich bei der Aufbereitung der Inhalte für statische RIAs um eine Erweiterung des jLAF-Frameworks handelt, sind die Inhalte in einer für das jLAF brauchbaren Weise abgelegt, d.h. die Texte liegen auf dem Server als HTML-Inhaltsfragmente vor (vgl. dazu Abschnitt 3.4.1.3 auf Seite 40), und einige Auflistungen werden zur Laufzeit bzw. – im Falle der Generierung der HTML-Fallback-Seiten – zur Generierungszeit aus Datenquellen bezogen.

Zu den aus Datenquellen zu beziehenden Informationen gehören:

- Aktuelle Meldungen
- Liste der betreuten Lehrveranstaltungen
- Publikationsverzeichnis
- Gästebucheinträge
- Fotoalben
- angebotene Videofilme

5.5.2.1 Webbook-Entities

Webbook-Entities sind konkrete Implementierungen der abstrakten Klasse `WebtoolEntity` (vgl. Abschnitt 3.5.4.1 auf Seite 66), die speziell für den Einsatz im Webbuch gedacht sind. Jede in der vorstehenden Aufzählung genannte Information wird in Form einer von `WebtoolEntity` abgeleiteten Klasse repräsentiert. Via Remoting sind diese Klassen immer zu erhalten. Abhängig davon, ob sie darüberhinaus aber auch als XML- oder als HTML-Dokument bezogen werden sollen, implementieren sie das (oder die) entsprechenden Interface(s). Das daraus resultierende Klassendiagramm zeigt Abbildung 5.7.

5.5.2.2 WebbookEntity-Manager-Klassen

Zur Verwaltung von Kollektionen von (Webbook-)Entites stehen jeweils einer Entität zugeordnete Verwalter-Klassen (Manager, vgl. Abschnitt 3.5.4.2 auf Seite 66) zur Verfügung. Abhängig davon, ob sie entsprechende Interfaces implementieren oder nicht, geben sie zu erkennen, ob sie die verwalteten Entitäten via XML-, HTML- und/oder Remoting-Transfer bereitstellen. Es ergibt sich daraus das in Abbildung 5.8 auf Seite 139 dargestellte Klassendiagramm.

Dem Diagramm ist zu entnehmen, dass die WebbookEntity-Manager-Klassen konkrete Implementierungen der abstrakten Klasse `WebtoolEntityManager` (siehe Abschnitt 3.5.4.2 auf Seite 66) sind. Ferner lässt das Klassendiagramm (aufgrund der Parameterbenennung `filename`) erkennen, dass die Daten, aus denen die Manager "ihre" Entity-Objekte instanziieren, jeweils

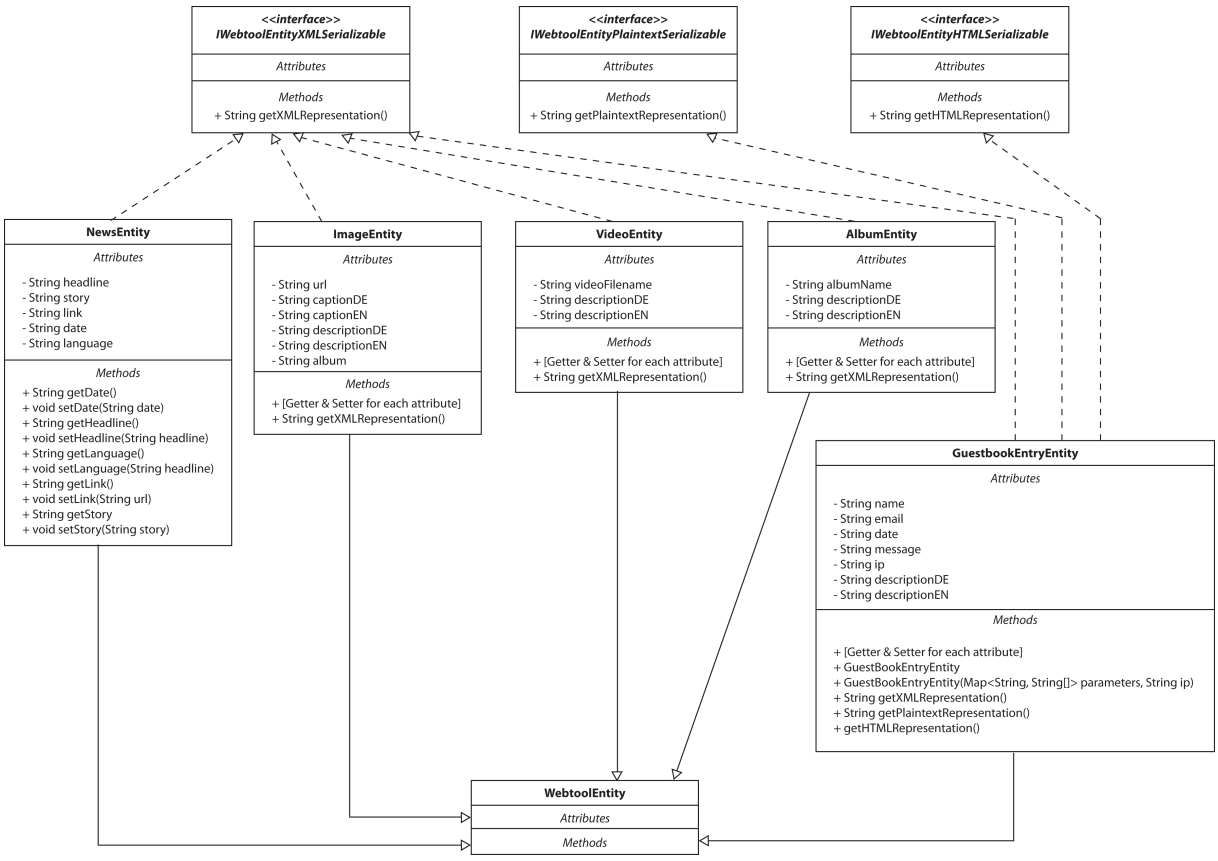


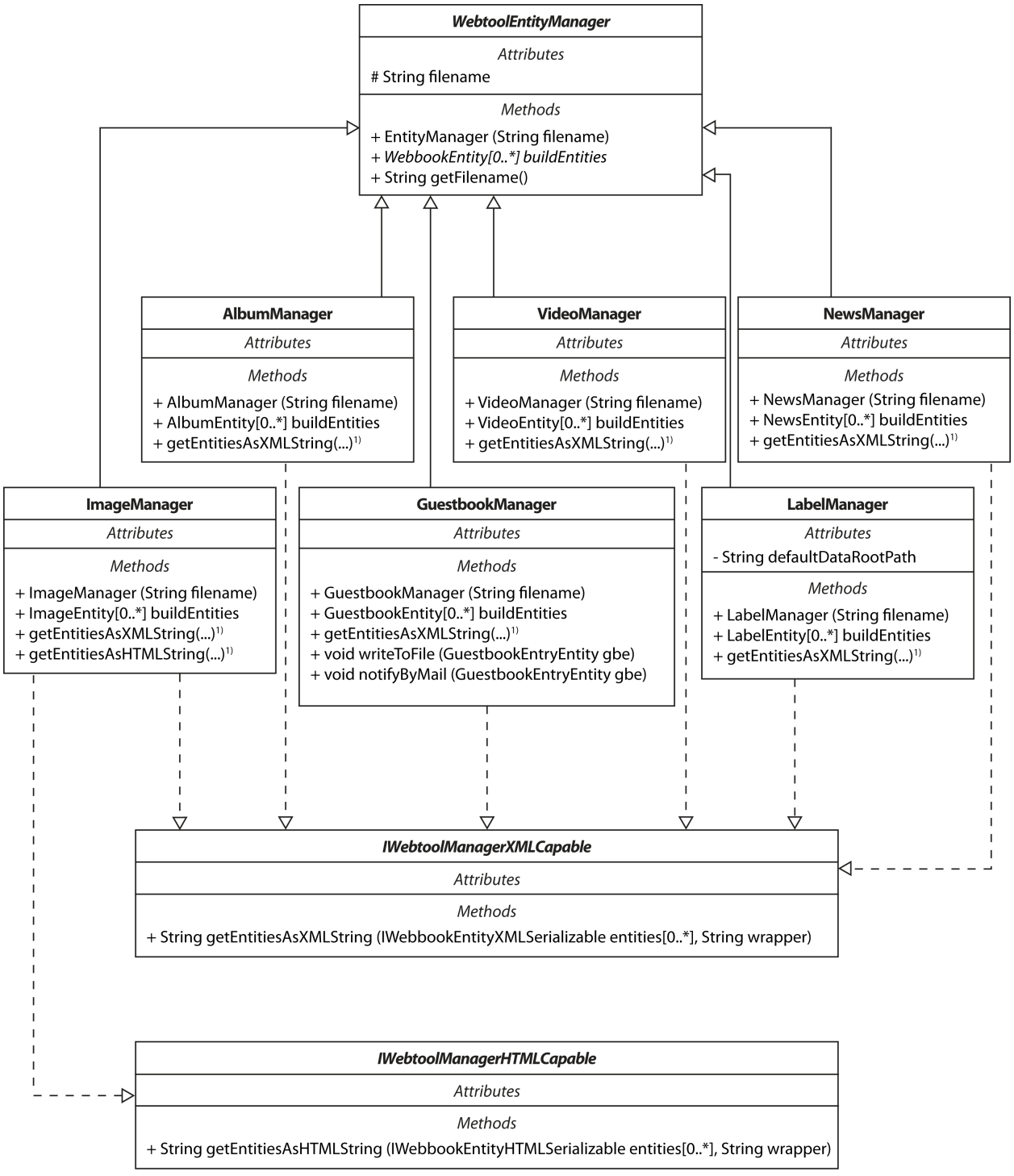
Abbildung 5.7: Die Implementierungen der Webtool-Entities im Webbook

aus einer Datei kommen. Dies ist hier nur als eine Möglichkeit von vielen zu sehen: ebensogut ist eine weitere Methode in den Klassen vorstellbar, welche es erlaubt, als Parameter eine Verbindung zu einer Datenbank oder zu einer sonstigen Quelle anzugeben und infolgedessen die benötigten Informationen aus einer anderen Quelle als einer Datei bezieht.

5.5.2.3 RAPs

Zum Zugriff von außen auf die Entitäten bzw. deren Manager stehen entsprechende *RemoteAccessPoints* (vgl. Abschnitt 3.5.5 auf Seite 72) zur Verfügung. Ihre Hierarche ist in Abbildung 5.9 auf Seite 140 gezeigt.

Die einzelnen Klassen des Frameworks, die in diesem sowie in den vorangegangenen Abschnitt(en) erläutert wurden, treffen in einer expliziten RAP-Klasse zusammen. Im Praxisbeispiel ist dies die in Listing 5.15 auf Seite 140ff. zitierte öffentliche Klasse `WebbookRAPs`. Anhand von Listing 5.15 kann studiert werden, welch einfache und übersichtliche Programmierung die Konzepte des serverseitigen jLAF erlauben.



1) Die Signatur wurde aus Grunden der Übersichtlichkeit nicht mit aufgeführt.

Abbildung 5.8: Klassendiagramm der Entity-Manager

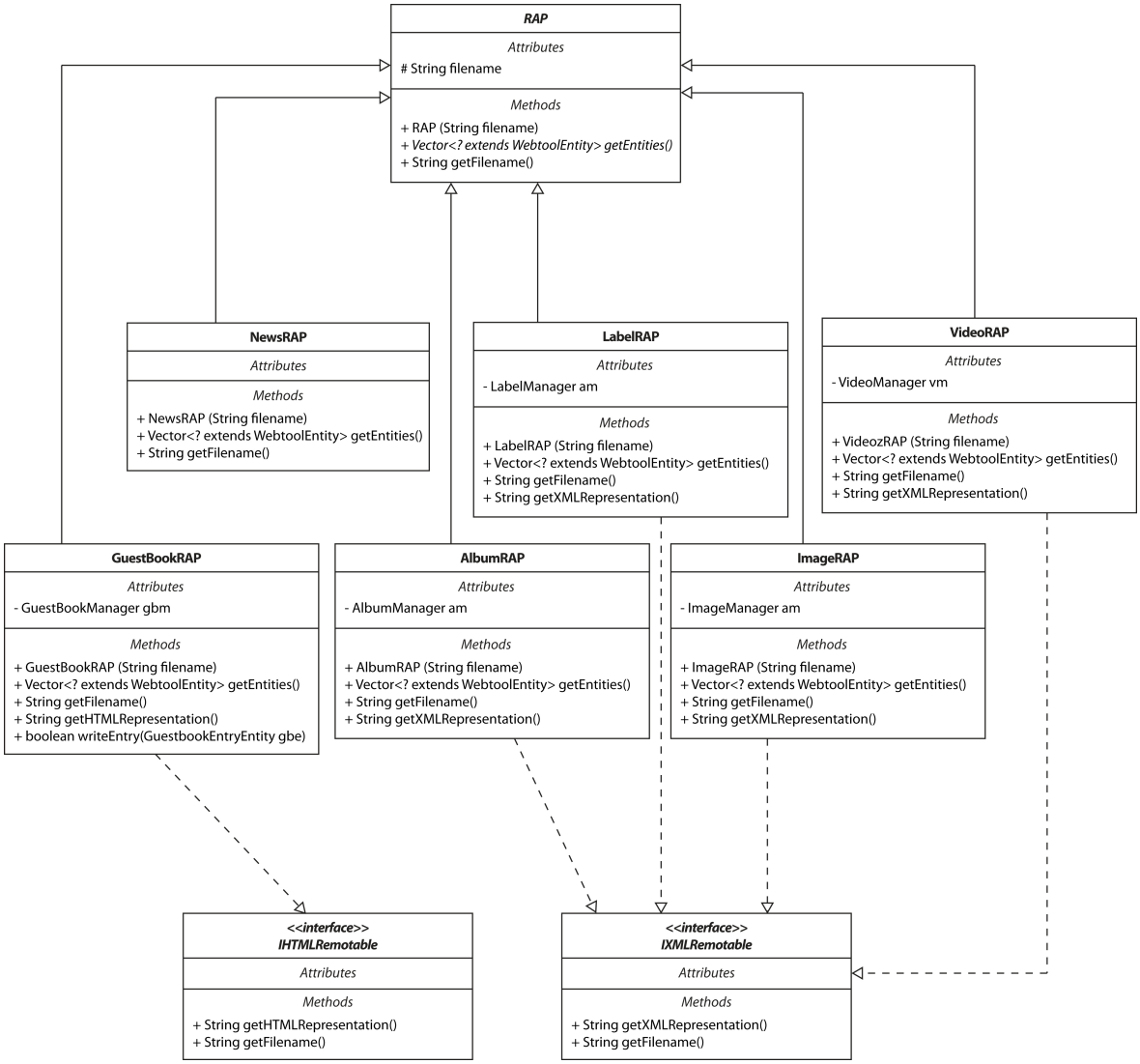


Abbildung 5.9: Konkrete Implementierungen der RAPs im Webbuch

Listing 5.15: Die RAPs des Webbuchs

```
1  // From all RAP classes, only this one is exposed to public access
2  public class WebbookRAPs {
3
4    // private methods, for internal use only
5
6    private Vector getEntitiesAsVector(RAP rap) {
7      return rap.getEntities();
8    }
9
10   private String getEntitesAsXML(IXMLRemotable xmlRAP) {
11     String xmlData = new String();
12     try {
13       xmlData = xmlRAP.getXMLRepresentation();
14     } catch (TransformException ex) {
15       xmlData = "### Transform exception while retrieving xml representation in
             file " + xmlRAP.getFilename() + ": " + ex;
16     } catch (WebbookException ex) {
17       xmlData = "### Webbook exception while retrieving album data for " +
```

```
            xmlRAP.getFilename() + ": " + ex;
18      }
19      return xmlData;
20    }
21
22    // public Remote Access Points (RAPs)
23
24    public Vector readGuestbook(String filename) {
25      return getEntitiesAsVector(new GuestBookRAP(filename));
26    }
27
28    public Vector getNews(String filename) {
29      return getEntitiesAsVector(new NewsRAP(filename));
30    }
31
32    public Vector getAlbums(String filename) {
33      return getEntitiesAsVector(new AlbumRAP(filename));
34    }
35
36    public Vector getImages(String filename) {
37      return getEntitiesAsVector(new ImageRAP(filename));
38    }
39
40    public Vector getVideos(String filename) {
41      return getEntitiesAsVector(new VideoRAP(filename));
42    }
43
44    public String getAlbumsAsXML(String filename) {
45      return getEntitesAsXML(new AlbumRAP(filename));
46    }
47
48    public String getImageDataAsXML(String filename) {
49      return getEntitesAsXML(new ImageRAP(filename));
50    }
51
52    public String getLabelsAsXML(String filename) {
53      return getEntitesAsXML(new LabelRAP(filename));
54    }
55
56    public String getVideosAsXML(String filename) {
57      return getEntitesAsXML(new VideoRAP(filename));
58    }
59  }
```

Erläuterungen zu Listing 5.15: Der Zugang zur Funktionsweise der RAP-Methoden gelingt am einfachsten über die öffentlichen Methoden, die ab Zeile 23 notiert sind. Es handelt sich – in Bezug auf den Methodenrumpf – jeweils um "Einzeiler", sodass die einfache und übersichtliche Handhabung der RAPs zu erkennen ist. Die Methoden in den Zeilen 23 bis 41 dienen zum Bezug von Collections (hier: Vektoren) bestimmter WebbookEntity-Klassen. Sie bekommen diese Entities jeweils durch die (private) Methode `getEntitiesAsVector` (Zeile 5 bis 7) geliefert. Analog dienen die Methoden ab Zeile 43 zum Erhalt der in einem XML-Dokument serialisierten Entity-Informationen. Auch besteht der Methodenrumpf lediglich aus dem Aufruf der in den Zeilen 9 bis 19 gezeigten privaten Methode `getEntitiesAsXML`. Zum besseren Verständnis des Ablaufs vom Aufruf einer RAP-Methode bis hin zum Erhalt des Rückgabewertes zeigen die Abbildungen 5.10 und 5.11 UML-Sequenz-Diagramme.

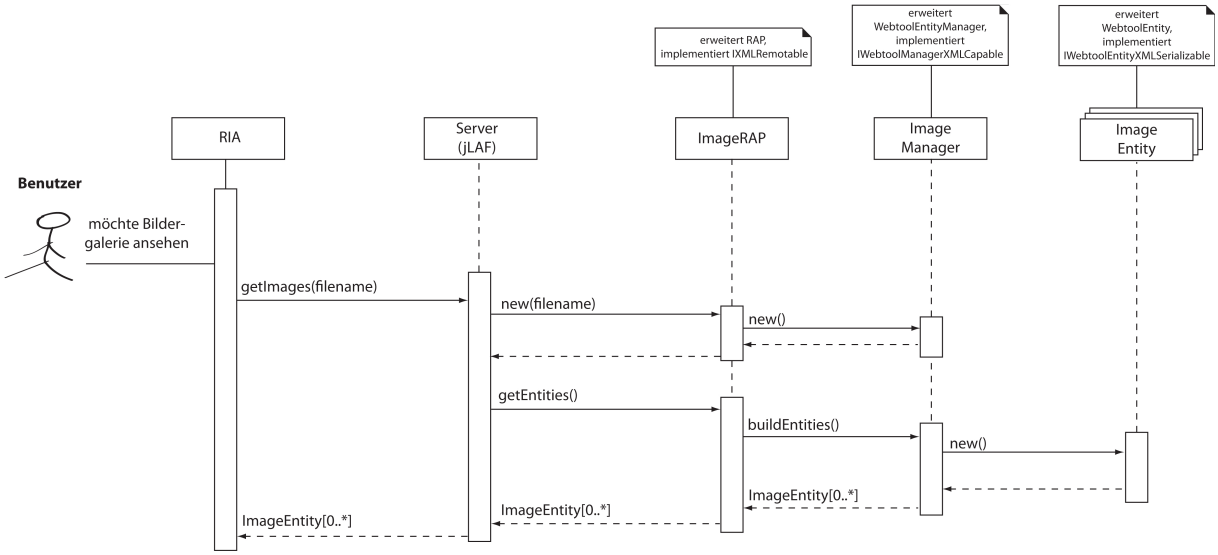


Abbildung 5.10: Sequenzdiagramm zur Funktionsweise des RAP-Patterns im Webbuch: Bezug von Bildinformationen via Remoting

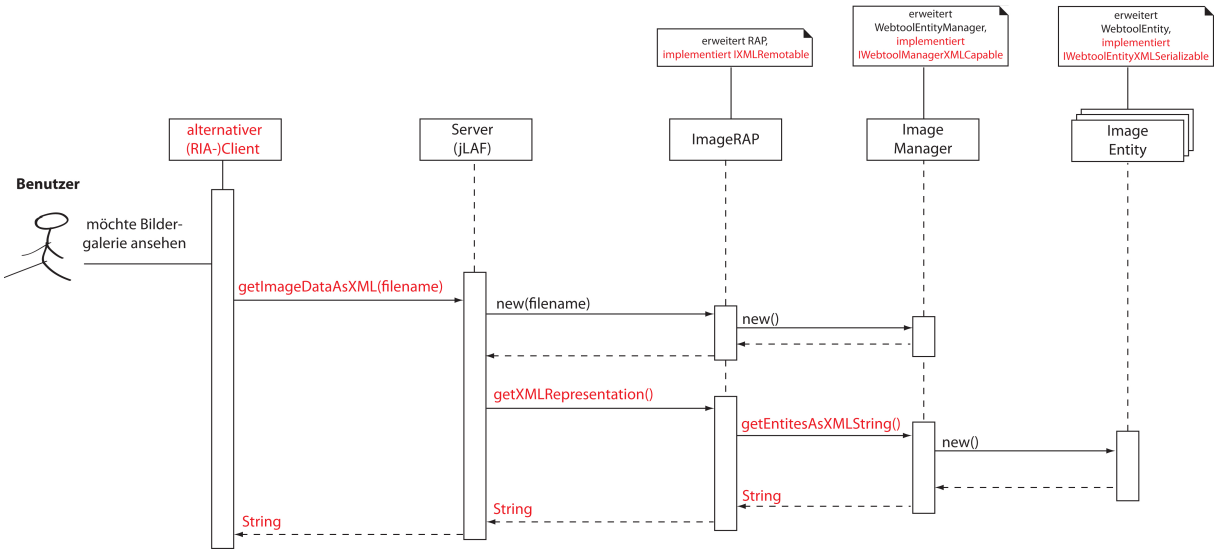


Abbildung 5.11: Sequenzdiagramm zur Funktionsweise des RAP-Patterns im Webbuch: Bezug von Bildinformationen als XML-Dokument (String)

Erläuterungen zu den Sequenzdiagrammen 5.10 und 5.11: Beide Diagramme visualisieren den Bezug von Bildinformationen (Titel, Unterschriften, Dateinamen), die clientseitig beispielsweise für das Erstellen einer Galerie weiterverwendet werden können (*ImageEntities*). Im Fall von Abbildung 5.10 ist der Bezug der Entities via *Remoting* dargestellt, und Abbildung 5.11 zeigt den Bezug der Bildinformationen als XML-Dokument. Die wesentlichen Unterschiede im Vergleich zum Remoting sowie die speziell für die Aufbereitung von XML-Dokumenten wichtigen Fakten sind dabei in roter Farbe gekennzeichnet. Analog zum Bezug der Daten via XML lässt sich auch die Übertragung der Daten in einer HTML-Aufbereitung zeichnen. Selbstverständlich gelten die Diagramme nicht nur für den Bezug von Bildinformationen, sondern in gleicher Weise auch für alle weiteren Entities im Webbuch (sofern diese die jeweiligen Interfaces implementieren).

Ein bestimmter Client wird in der Regel wohl nur eine Bezugsart verwenden. So erfolgt die Datenübertragung beispielsweise im Webbuch stets via Remoting. Ebenso ist aber auch ein Szenario denkbar, in dem sich verschiedene Clients mit den RAPs auf dem Server verbinden wollen, wobei die Clients durchaus unterschiedliche Bezugsarten erfordern können. Deshalb ist in Abbildung 5.11 die Bezeichnung "RIA" in "alternativer (RIA-)Client" umgeändert worden: dies soll anzeigen, dass sich hier ein anderer Client als in Abbildung 5.10 verbindet, nämlich einer, der die Bildinformationen in einer XML-Aufbereitung benötigt. In der weiteren Beschreibung dieses Praxisbeispiels wird noch einmal auf die HTML-Version der Bildergalerien im Webbuch Bezug genommen (siehe Abschnitt 5.5.5.1 auf Seite 148). Dann würde das an die HTML-Übertragung angepasste (hier aufgrund seiner Analogie zu Abbildung 5.11 nicht mehr dargestellte) Sequenzdiagramm zur Anwendung kommen.

Von allen im Framework enthaltenen Klassen bzw. den eigenen Implementierungen der abstrakten Klassen wird lediglich die in Listing 5.15 gezeigte Klasse öffentlich gemacht. Alle anderen erwähnten Klassen und Methoden dienen nur zur internen Verwendung. Im Falle des auf Flex und BlazeDS basierenden Praxisbeispiels ist "öffentlich machen" gleichbedeutend mit der Freigabe der Klasse zum Zugriff via Remoting. Listing 5.16 zeigt die hierzu notwendige Ergänzung in der BlazeDS-Konfigurationsdatei `remoting-config.xml`. Nach dem Hinzufügen dieses Eintrags (Zeilen 5 bis 11) in die besagte Datei und einem Neustart des Servers können die RAP-Methoden vom Flex-Client aus verwendet werden.

Listing 5.16: Konfiguration der Webbook-RAP-Klasse in BlazeDS

```
1  <service>
2
3    <!-- other content... -->
4
5    <destination id="webbookRemoting">
6      <properties>
7        <source>
8          de.norbertheidenbluth.webbook.remoteAccessPoints.WebbookRAPs
9        </source>
10     </properties>
11   </destination>
12
13 </service>
```

5.5.3 Clientseitige Implementierung

Wie in diesem Kapitel erklärt wurde, sind auch clientseitige Implementierungen notwendig, um die beschriebenen Konzepte verwenden zu können. Derartige Implementierungen liegen auch

im Webbuch vor. Ihre Umsetzung lässt sich am besten anhand eines Codebeispiels zeigen, welches für diesen und den folgenden Unterabschnitt nun der Betrachtungsgegenstand sein soll (Listing 5.17).

Listing 5.17: Abstrakte Klasse `NorbertsPage.as` als Beispiel für die Individualisierbarkeit

```
1  public class NorbertsPage extends Page implements IMultilingual
2  {
3    protected var _styleSheet: StyleSheet;
4    protected var _captionFile: String;
5    protected var _cssFileName: String;
6
7    [Bindable]
8    protected var _mll: MultiLanguageLabeler;
9
10   public function NorbertsPage() {
11     super();
12     setStyle("backgroundAlpha", "1.0");
13     _captionFile = null;
14     Application.application.addEventListener(LanguageChangeEvent.
           LANGUAGE_CHANGED, onLanguageChange);
15     addEventListener(FlexEvent.CREATION_COMPLETE, onCC);
16   }
17
18   private function setLabels() : void {
19     initialize();
20   }
21
22   protected function updateStyle() : void {
23     // To be implemented by subclasses
24   }
25
26   private function onCC(e: Event) : void {
27     _mll = new MultiLanguageLabeler(_captionFile, Application.application.
           session.language);
28     additionalOnCC();
29   }
30
31   protected function additionalOnCC() : void {
32     // To be implemented by subclasses if necessary
33   }
34
35   public function onLanguageChange(lce: LanguageChangeEvent) : void {
36     if (_captionFile == null) { return; }
37     _mll.setLanguage(lce.language);
38     setLabels();
39     additionalOnLanguageChange();
40   }
41
42   protected function additionalOnLanguageChange() : void {
43     // To be implemented by subclasses
44   }
45
46   private function onGetLabels(r: ResultEvent) : void {
47     setLabels();
48     additionalOnGetLabels();
49   }
50
51   protected function additionalOnGetLabels() : void {
52     // To be implemented by subclasses if necessary
53   }
54 }
```

Erläuterungen zum Codebeispiel 5.17: Jede virtuelle Seite des RIA-Webbuchs ist eine konkrete Implementierung der in Listing 5.17 vollständig gezeigten abstrakten Klasse `NorbertsPage`[21]. Einige Methoden dieser Klasse sind ausprogrammiert und als *private* gekennzeichnet. Sie stehen somit in abgeleiteten (bzw. implementierenden) Klassen zum Aufruf zur Verfügung, dürfen dort aber nicht überschrieben werden. Dies trifft insbesondere auf die Methoden `onCC`, `onLanguageChange` und `onGetLabels` zu. Wie die Nomenklatur dieser Methoden bereits anzeigt, handelt es sich hierbei um Ereignisbehandlungsmethoden, welche genau dann aufgerufen werden, wenn die Seite initialisiert wurde (Methode `onCC`[22]), die Sprache geändert wurde (Methode `onLanguageChange`) oder die aus externen Quellen bezogenen Sprachinformationen eingetroffen sind (Methode `onGetLabels`). Durch das Unterbinden des Überschreibens dieser Methoden in den Unterklassen wird sichergestellt, dass diese Unterklassen stets ein definiertes (Mindest-)Verhalten zeigen, wenn sie über das Eintreten eines entsprechenden Ereignisses informiert werden. *Zusätzlich* können die abgeleiteten Klassen noch ein eigenes, individuelles Verhalten ergänzen, indem sie die leeren Methoden `additionalOnCC`, `additionalOnLanguageChange` und `additionalOnGetLabels` ausfüllen, denn diese Methoden dürfen – wie ihr Zugriffsmodifikator *protected* anzeigt – überschrieben werden.

5.5.3.1 Zugriffsmechanismen

Im Falle des Webbuchs liegt mit der in Abbildung 5.3 auf Seite 119 gezeigten Klasse `WebbookRemotingConnector` bereits eine Implementierung eines `RemotingConnectors` vor, die auf die Kommunikation mit dem in Zeile 5 des Listings 5.16 benannten Service-Endpunktes vorkonfiguriert ist. Entsprechend gibt es mit der Klasse `WebbookServletConnector` eine auf den Einsatz im Webbuch spezialisierte Ausprägung der Klasse `ServletConnector` zum Zugriff auf Webbuch-Servlets. Die konkreten Implementierungen dieser beiden Klassen sind für das hier vorgestellte Gesamtkonzept jedoch nicht von Bedeutung, sodass auf ihre detailliertere Beschreibung verzichtet werden soll.

5.5.3.2 Clientseitige DAO-Klassen

Wenn ein Remoting-Mechanismus zum Einsatz kommen soll, müssen den serverseitigen DAO-Klassen lokale (d.h. clientseitige) Entsprechungen zugeordnet werden. Im Falle der im Webbuch verwendeten BlazeDS-Technologie erfolgt diese Zuordnung durch zwei Annotationen, die in die clientseitigen Klassen aufgenommen werden. Listing 5.18 zeigt ein Beispiel dafür.

Listing 5.18: Annotationen zur Zuordnung einer clientseitigen Klasse zu ihrer Entsprechung auf der Serverseite

```
1 package de.norbertheidenbluth.webbook.dao {
2   [Bindable]
3   [RemoteClass(alias="de.norbertheidenbluth.webbook.entities.ImageEntity")]
4   public class ImageEntity {
5     // implementation omitted
6   }
7 }
```

[21] Flex unterstützt aktuell keine abstrakten Klassen, sodass es sich bei dem gezeigten Code-Beispiel technisch gesehen nicht um eine abstrakte Klasse handelt, weil zu jeder Methode eine (leere) Implementierung vorliegt und deshalb Objekte dieser Klasse instanziiert werden können. *De facto* hat diese Klasse aufgrund der leeren Implementierungen aber die Bedeutung einer abstrakten Klasse, sodass erst Objekte der abgeleiteten Klassen wirklich sinnvoll verwendet werden können.

[22] Dabei steht die Abkürzung `CC` für *Creation Complete[d]*.

Erläuterungen zu Listing 5.18: Das Listing zeigt die Kopfzeilen der Definition der Klasse `ImageEntity`. Die konkrete Implementierung dieser Klasse ist hier unbedeutend und aus diesem Grund nicht gezeigt (siehe Kommentar in Zeile 5). Wesentlich sind stattdessen die Zeilen 2 und 3 des Listings. Sie geben an, dass die aktuell definierte Klasse eine Entsprechung auf dem entfernten Server in Form der dortigen Klasse `ImageEntity` des Pakets `de.norbertheidenbluth.webbook.entities` besitzt.



Abbildung 5.12: Zusammenhang zwischen den DAO-Klassen auf Client- und Serverseite

Abbildung 5.12 fasst den Zusammenhang zwischen den clientseitigen DAO-Klassen und ihren serverseitigen Pendants am Beispiel der DAO-Klasse `ImageEntity` nochmals bildlich zusammen[23].

5.5.4 Zur Individualisierbarkeit des Webbuchs

5.5.4.1 Layout/CSS

Aufgrund der vollständigen Trennung von Layout und Inhalt ist es prinzipiell möglich, jedes beliebige Layout zur Präsentation zu verwenden. In Zusammenhang mit dem Webbuch ist jedoch interessant, dass dem Benutzer die Möglichkeit gegeben werden soll, innerhalb des vorgegebenen Rahmens eigene Layout-Präferenzen zu wählen.

Zu diesem Zweck stehen zu Demonstrationszwecken drei Layout-Variationen zur Verfügung, die selbstverständlich durch eigene ergänzt werden können. Die einzelnen Layouts werden in

[23] Die in Abbildung 5.12 erkennbare Verwendung öffentlicher Membervariablen auf der Serverseite ist kein Resultat nachlässiger Programmierung, sondern Voraussetzung für die Verwendbarkeit in Zusammenhang mit BlazeDS.

einer externen, speziell für Flex-Komponenten angepassten Form einer CSS-Datei vorgehalten und haben folgende Ausprägungen:

Standard: das Standard-Layout (das auch im Screenshot in Abbildung 5.6 gezeigt ist),

Hoher Kontrast: ein Layout, das – insbesondere unter Verzicht auf Hintergrundgestaltungen – einen besonders hohen Kontrast zwischen Vorder- und Hintergrund bietet,

Große Schriften: ein Layout, das auf bessere Lesbarkeit durch den Einsatz besonders großer Schriften ausgerichtet ist; da der zur Verfügung stehende Platz für die Texte aber nicht gleichzeitig mit dem Schriftgrad wächst, sondern konstant bleibt, wird der zusätzlich benötigte Platz dadurch gewonnen, dass der Text innerhalb seines vorgesehenen Rahmens ein Verschieben ("*scrollen*") erfordert.

Entsprechend den Ausführungen in Abschnitt 5.3.4 auf Seite 127 ist eine weitere visuelle Anpassung des Webbuchs an die Benutzerpräferenzen jederzeit problemlos realisierbar.

5.5.4.2 Zur Unterstützung der Mehrsprachigkeit im Webbuch mit Hilfe des MultiLanguageLabelers

Das Webbuch unterstützt Mehrsprachigkeit durch Verwendung des MLL-Patterns. Jede konkrete Implementierung der Klasse `NorbertsPage` (siehe Listing 5.17 auf Seite 144) verfügt über die Angabe, in welcher Datei ihre Beschriftungen zu finden sind (Membervariable `captionFile` in Zeile 4). Ferner erben die Implementierungen ein zur Datenbindung taugendes `MultiLanguageLabeler`-Objekt (Zeilen 7 und 8). Dieses Objekt wird nach erfolgreicher Initialisierung der Seiten mit der aktuell eingestellten Sprache konfiguriert (Zeile 19). Die Seiten-Objekte überwachen das Ereignis `LanguageChangedEvent` (Zeile 14), das immer dann ausgelöst wird, wenn der Benutzer eine Änderung der Sprache vornimmt. Die Behandlung dieses Sprachänderungsereignisses übernimmt zunächst die private (also nicht in abgeleiteten Klassen veränderbare) Methode `onLanguageChanged` (Zeilen 39 bis 46), wobei in Zeile 45 ein Aufruf der Methode `additionalOnLanguageChange` erfolgt, durch welchen die konkreten Unterklassen von `NorbertsPage` noch eine *zusätzliche* Behandlung des Sprachänderungsereignisses vornehmen können.

5.5.5 Zur Generierung der HTML-Fallback-Lösung des Webbuchs

Die Seiten des Webbuchs, auf denen Fließtext vorgesehen ist, beziehen diesen aus denselben HTML-Inhaltsfragment-Seiten, die auch Grundlage für die Generierung der HTML-Fallback-Lösung sind. Insbesondere werden die Texte demnach nicht zur Design-Zeit eingebunden, sondern erst zur Laufzeit vom jLAF-Framework bezogen. (Für diese Operation steht eine eigene RAP-Methode zur Verfügung, die hier aber nicht weiter von Bedeutung ist.)

Die Erzeugung der HTML-Fallback-Version des Webbuchs entspricht somit im Wesentlichen der Generierung von Webauftritten, wie sie in Kapitel 3 in Zusammenhang mit dem Webtool bzw. dem jLAF-Framework erläutert wurde. Um Wiederholungen zu vermeiden, greifen die nachfolgenden Erklärungen nur die Bereiche auf, die sich entweder von den Ausführungen in Abschnitt 3.5 unterscheiden oder diese ergänzen. Ansonsten sei zwecks Erklärung auf Kapitel 3 verwiesen.

Somit ist die beabsichtigte "doppelte" Verwendbarkeit des jLAF gelungen: einerseits beliefert das Framework eine (statische) RIA, zugleich ermöglicht es aber auch die automatische Generierung der HTML-Fallback-Version dieser RIA.

5.5.5.1 Ein Beispiel für Graceful Degradation: Bildergalerien im Webbuch

Bestandteil des Webbuchs sind Bildergalerien, in denen – thematisch sortiert – selbstgemachte Fotos betrachtet werden können. Zur optischen Auflockerung – und um den Besucher neugierig(er) zu machen – werden die Galerien nicht "nur dargestellt" sondern durch den sogenannten *CoverFlow*-Effekt, der durch Apples iTunes-Software populär geworden ist und durch einige Animationen die *User Experience* beim Betrachten der Bilder erhöht, präsentiert. Neben den Bildern werden auch erläuternde Unterschriften (ein Titel und eine Beschreibung) angezeigt. Diese können in verschiedenen Sprachen vorliegen, um ihrerseits einen Beitrag zur Mehrsprachigkeit zu leisten. Einen Screenshot der Bildergalerie zeigt Abbildung 5.13.

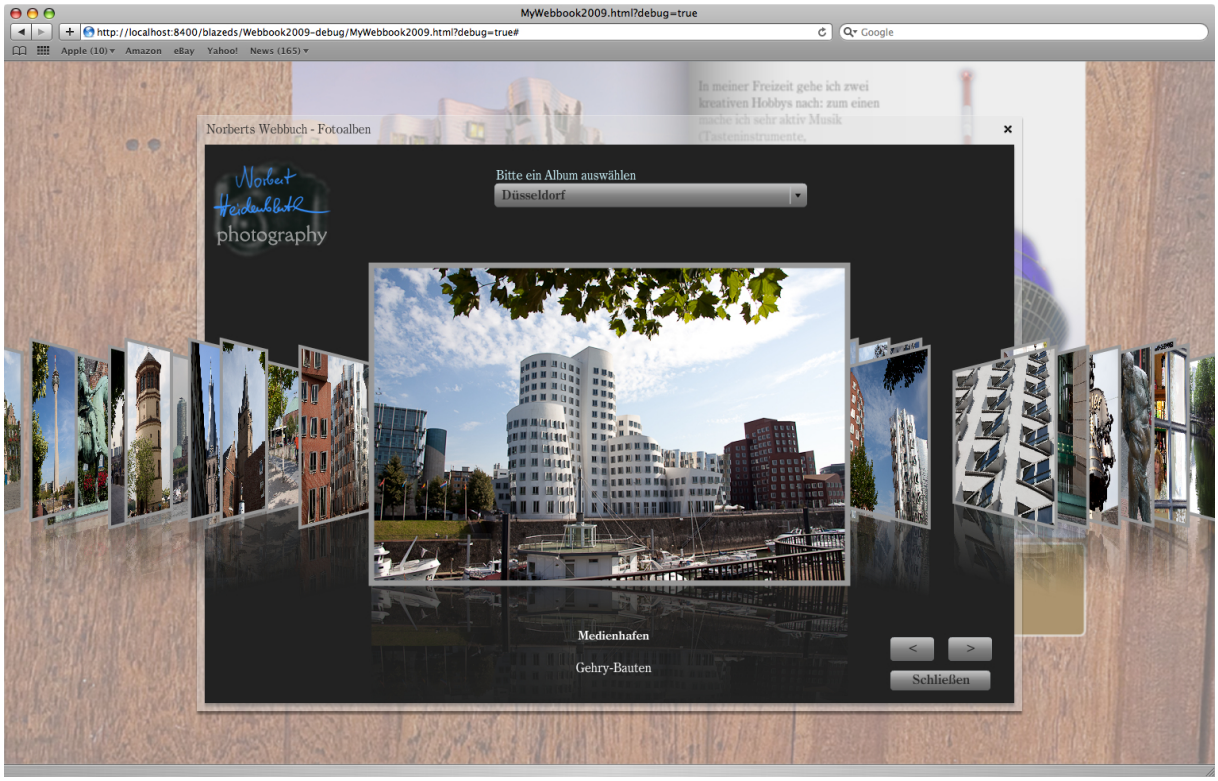


Abbildung 5.13: Screenshot der Bildergalerie im Webbuch

Die Bildinformationen (Dateinamen, Unterschriften und Titel in den verschiedenen Sprachen) werden sowohl auf der Client- als auch auf der Serverseite als (Webbook-)Entity-Objekt geführt. Anhand der Klasse `ImageEntity` wurden in Abbildung 5.12 auf Seite 146 die Implementierungen bereits in anderem Zusammenhang gezeigt. Während die Client-Implementierung dieser Klasse lediglich die Membervariablen deklariert, implementiert die Serverversion zudem noch die Interfaces `IWebtoolEntityXMLSerializable` und `IWebtoolEntityHTMLSerializable`, wobei die Implementierung des letzteren anzeigt, dass die Klasse sich auch als HTML-Quellcode-Fragment zu repräsentieren weiß. Diese Fähigkeit wird verwendet, um eine HTML-Fallback-Lösung auch für die Bildergalerien zu generieren. Letztere soll ebenfalls eine bessere *User Experience* bieten, als es eine simple Darstellung der Bilder machen würde, zugleich aber

selbstverständlich barrierefrei sein und sich durch Beachtung der Paradigmen *Graceful Degradation* und *Unobstrusive JavaScript* (vgl. Abschnitte 5.4.1.2 und 5.4.4) an alle Ausgabegeräte anpassen können.

Zu diesem Zweck findet in der HTML-Fallback-Lösung das JavaScript-Tool *Lightbox* von Lokesh Dhakar[24] Verwendung. Es bietet eine ansprechendere Form der Bildpräsentation, als dies mit reinem HTML möglich ist. Wenn das verwendete Ausgabegerät JavaScript *nicht* unterstützt, trifft es jedoch auf validen und brauchbaren HTML-Code zur traditionellen Anzeige von Bildern mittels HTML. Deshalb ist die HTML-Repräsentierung der `ImageEntity`-Objekte darauf ausgerichtet, in Zusammenhang mit dem Script *Lightbox* verwendet zu werden.

Die beschriebenen einzelnen Komponenten ergeben nun den folgenden Gesamtzusammenhang:

1. Die RIA bezieht und verwaltet Informationen über die zu einer Galerie zusammengefassten Bilder mit Hilfe der server- und clientseitig vorhandenen Klassen `ImageEntity`. Unter Verwendung des animierten CoverFlow-Effekts können die Bilder in der RIA betrachtet werden.

2. Zur automatischen Generierung des HTML-Fallback-Albums wird dieselbe Datenbasis verwendet, die auch in der RIA zum Einsatz kommt. Ebenso werden die Bildinformationen serverseitig auf exakt dieselbe Weise verwaltet, wie dies der Fall ist, wenn sie zur Verwendung in der RIA-Fall angefragt werden, nämlich in Form von *Collections* von `ImageEntity`-Objekten.

3. Um eine HTML-Repräsentierung zu ermöglichen, implementieren die `ImageEntity`-Objekte (unter anderem) das Interface `IWebtoolEntityHTMLSerializable`.

4. Die Methode `getHTMLRepresentation`, welche durch das Interface `IWebtoolEntityHTMLSerializable` gefordert wird, ist dabei auf die Verwendung innerhalb des JavaScript-Tools *LightBox* konzipiert. Die Implementierung dieser Methode zeigt Listing 5.19.

5. Ein HTML-Inhaltsfragment dient zur thematischen Hinführung auf eine Galerie. Dieses Fragment enthält eine Direktive, mit der die `ImageEntity`-Objekte erzeugt und in HTML-Code transformiert werden. Das Ergebnis dieser Code-Erzeugung ersetzt den Direktiven-Aufruf im HTML-Inhaltsfragment und führt so auf eine vollständige HTML-Seite, die auf einem Ausgabegerät zur Anzeige gebracht werden kann.

6. Die generierte HTML-Seite verwendet mit *LightBox* zwar eine JavaScript-Bibliothek, wenn ein Ausgabegerät jedoch nicht in der Lage ist, JavaScript-Code zu interpretieren, "sieht" das Ausgabegerät validen HTML-Code (vgl. Codebeispiel 5.20), welcher dann zur Ausgabe einer zwar stark vereinfachten, jedoch noch brauchbaren Version verwendet wird.

Listing 5.19: Implementierung der Methode `getHTMLRepresentation()` zur Verwendung mit dem JavaScript-Tool *LightBox*

```
1 public String getHTMLRepresentation() {
2   return "<a href=\"" + url + "\" rel=\"lightbox[" + album +
3     "]\" title=\"" + caption + "\">" + caption + "</a>\n";
4 }
```

[24] siehe `http://www.huddletogether.com/projects/lightbox2/`

Listing 5.20: HTML-Code der Bildergalerie: Ein Beispiel für unobstrusive JavaScript

```
1 ...
2 <a href="img/staedte/duesseldorf/ddorf16.jpg" rel="lightbox[staedte/duesseldorf]
    " title="Medienhafen">Medienhafen</a>
3 <a href="img/staedte/duesseldorf/ddorf17.jpg" rel="lightbox[staedte/duesseldorf]
    " title="Wasserspiel am Landtag">Wasserspiel am Landtag</a>
4 <a href="img/staedte/duesseldorf/ddorf18.jpg" rel="lightbox[staedte/duesseldorf]
    " title="Glocke am Brauhaus "Uerige"">Glocke am Brauhaus "Uerige"</a>
5 <a href="img/staedte/duesseldorf/ddorf19.jpg" rel="lightbox[staedte/duesseldorf]
    " title="Typische Duesseldorfer Laterne">Typische Duesseldorfer Laterne</a>
6 <a href="img/staedte/duesseldorf/ddorf19b.jpg" rel="lightbox[staedte/duesseldorf
    ]" title="Skulptur 'Die Auseinandersetzung'">Skulptur "Die
    Auseinandersetzung"</a>
7 <a href="img/staedte/duesseldorf/ddorf20.jpg" rel="lightbox[staedte/duesseldorf]
    " title="Schnapsladen 'Killepitsch'">Schnapsladen "Killepitsch"</a>
8 <a href="img/staedte/duesseldorf/ddorf21.jpg" rel="lightbox[staedte/duesseldorf]
    " title="Koe–Graben">Koe–Graben</a>
9 ...
```

Die vorstehende Aufzählung zeigt, wie mit Hilfe der bislang vorgestellten Konzepte die an der Maximierung der *User Experience* ausgerichtete RIA-Version der Bildergalerie (im Webbuch) sukzessive über eine JavaScript-Fassung bis hin zu einer einfachen HTML-Alternativ-Text-Version im Sinne der *Graceful Degradation* "degradiert". Dabei steht mit Punkt 5 eine HTML-Fallback-Lösung zur Verfügung, die – erkennbar an Punkt 6 – eine "unaufdringliche", barrierefreie Verwendung von JavaScript (*Unobstrusive JavaScript*) vorsieht. Wenn Bilder überhaupt nicht mehr dargestellt werden können (insbesondere beim Einsatz von Screenreadern), dann informiert das Attribut `title` zumindest textuell über den Inhalt, der durch die Bilder vermittelt wird, und dient mithin als Alternativ-Text.

So, wie es im Beispiel anhand von Bildinformationen erklärt ist, verfährt das Webbuch auch mit anderen Klassen aus der `WebbokEntity`-Hierarchie. Beispielsweise werden aktuelle Mitteilungen auf eine vergleichbare Weise über den "Umweg" eines JavaScript-Tools aufbereitet, welches ebenfalls "unaufdringlich" ist und insoweit auf einen HTML-Code führt, der auch dann noch eine sinnvolle Aufbereitung der Nachrichten garantiert, wenn weder die RIA- noch die JavaScript-Versionen verwendet werden können, ansonsten aber aufgrund einer "Anreicherung" mit (unaufdringlichem) JavaScript einen Kompromiss zwischen der reichhaltigen Darstellung in der RIA-Version und der "spartanischen" reinen HTML-Variante repräsentiert.

5.5.6 Zusammenfassung des Praxisbeispiels

Das Praxisbeispiel "Webbuch" setzt die in diesem Kapitel vorgestellten Konzepte in einen Gesamtzusammenhang, der zugleich die Praxistauglichkeit der beschriebenen Lösungen unter Beweis stellt. Im Sinne der *Graceful Degradation* erhält man aus der RIA-Version eines (inhaltszentrierten) Webauftritts sukzessive eine barrierefreie HTML-Fallback-Lösung.

Einer solchen Fallback-Lösung kommen dabei zwei gewichtige Bedeutungen zu: einerseits dienen sie als – je nach Verwendungszweck der Seite gesetzlich geforderte – barrierefreie Fassung einer ansonsten nicht auf Barrierefreiheit bedachten RIA-Version. Andererseits sind sie auch von Suchmaschinen auffindbar und indizierbar, sodass es letztlich die HTML-Versionen und nicht die RIAs sind, die zur Auffindbarkeit der Seiten durch Suchmaschinen führen.

Das jLAF-Framework und das Praxisbeispiel enthalten bereits Lösungsmöglichkeiten, um Individualisierungen der RIAs zur Laufzeit zu ermöglichen. So sollen Benutzer z.B. die ihnen vertrauteste Sprache und ein ihnen angenehmes Layout individuell und zur Laufzeit wählen

können. Weitergehende Individualisierungsansätze lassen sich an den in diesem Kapitel betrachteten statischen RIAs nicht ausmachen. Dies ändert sich jedoch, wenn man anstelle der *statischen* RIAs die *dynamischen* RIAs betrachtet. Diese bergen ein deutlich höheres Potenzial für Individualisierungen. Frage- und Problemstellungen zu diesem Themenbereich sind – zusammen mit entsprechenden Lösungsansätzen – der Gegenstand der restlichen Kapitel dieser Arbeit.

Kapitel 6

Strategien zur Laufzeit-Personalisierung dynamischer RIAs

"With a little bit of forethought,
you can build an application
that can be used by everybody
whilst giving the majority
an enhanced experience."[Kei]

Im restlichen Teil dieser Arbeit stehen nicht mehr die *statischen*, sondern stattdessen die *dynamischen* RIAs im Mittelpunkt der Betrachtung. Erneut werden dabei jedoch nur die RIAs der Kategorie 2 (gemäß der Definition auf Seite 83) betrachtet, nicht hingegen JavaScript-basierte RIAs. Somit liegt der Fokus nun auf der Eigenschaft dieser Klasse von RIAs, eine vollwertige Anwendung (*Application*) zu sein.

Dynamische RIAs sind also "hybrid": einerseits sind sie Webseiten, andererseits aber auch Anwendungen. Dieser hybride Charakter führt auf spezifische Problemstellungen: dadurch, dass sie als Webseite auftreten, gelten für sie (je nach Einsatzszenario) die zu Beginn dieser Arbeit erläuterten gesetzlichen Anforderungen an ihre Barrierefreiheit. Darüberhinaus ergeben sich Fragestellungen aber auch aus dem von dynamischen RIAs auszuübenden "Spagat", sowohl eine Web-Anwendung als auch eine Desktop-Applikation zu sein (bzw. sein "zu wollen"). Diese Fragen lassen sich nicht mit Ansätzen aus der "reinen" Web-Programmierung bzw. der "reinen" Anwendungsentwicklung beantworten. Vielmehr erfordern sie, neue Lösungen zu finden, die ihrem hybriden Charakter unmittelbar gerecht werden.

Drei ausgewählten Problemstellungen dieser Natur widmet sich das vorliegende Kapitel, indem die Probleme zunächst vorgestellt und sodann mögliche Lösungen dazu vorgeschlagen werden. Dies geschieht in diesem Kapitel in einer abstrakten, unabhängig von einem konkreten Anwendungsfall bleibenden Form und führt zu drei Entwurfsmustern (*Design Pattern*). Somit soll die Allgemeingültigkeit der vorgeschlagenen Lösungen und deren Wiederverwertbarkeit zum Ausdruck gebracht. Eine eigene Integration der Pattern in ein konkretes Software-Projekt und somit der Beweis der Praxistauglichkeit der vorgeschlagenen Lösungen sind Gegenstand des nachfolgenden Kapitels.

Im Unterschied zu den vorangegangenen Kapiteln wird der Themenkomplex Barrierefreiheit nun nicht mehr isoliert betrachtet, sondern als eine Säule in einem Gesamtkonzept zur "Individualisierbarkeit" von RIAs verstanden. Insofern beschäftigt sich diese Arbeit fortan mit der Frage, ob und ggf. wie dynamische RIAs eine Individualisierbarkeit durch den Benutzer unterstützen können.

6.1 Zur Barrierefreiheit, Individualisier- und Personalisierbarkeit von RIAs

6.1.1 Dynamische RIAs und Barrierefreiheit

Dynamische RIAs lassen sich nicht mit Hilfe des bisher vorgestellten Frameworks jLAF in eine HTML-Fallback-Version überführen, denn dieses Framework wurde für statische Inhalte konzipiert. Die bei dynamischen RIAs unverzichtbare clientseitige Logik lässt sich nicht mittels jLAF in HTML-Versionen überführen. Aus den Ausführungen des Abschnitts 4.5 auf Seite 97ff. geht hervor, dass RIAs der Kategorie 2 durchaus jeweils eigene Ansätze zur Unterstützung von Barrierefreiheit – vor allem alternativer Ausgabegeräte wie Screenreader – mitbringen. Jedoch obliegt es hier stets dem Entwickler, für die korrekte Umsetzung der Ansätze zu sorgen. Eine automatische Generierung einer vereinfachten Version einer dynamischen RIA (z.B. als JavaScript-Variante anstelle der Flash-Version) ist zwar durchaus vorstellbar (OpenLaszlo nimmt diesen Gedanken bereits auf, vgl. dazu Abschnitt 4.5.4), sie ist aber nicht Gegenstand der vorliegenden Arbeit.

6.1.2 Zum Begriff der Individualisierbarkeit

In der ISO-Norm 9241-110 wird in Abschnitt 4 ("Grundsätze der Dialoggestaltung und zugehörige Empfehlungen") im einleitenden Überblick die *Individualisierbarkeit* als eines von sieben grundsätzlichen Zielen bei der Gestaltung und Bewertung eines Dialogs genannt. Zur Definition des Begriffs wurde in der Norm die folgende Formulierung verwendet:

> *"Ein Dialog ist individualisierbar, wenn Benutzer die Mensch-System-Interaktion und die Darstellung von Informationen ändern können, um diese an ihre individuellen Fähigkeiten und Bedürfnisse anzupassen."*

In Anmerkung 2 zu dieser Definition heißt es weiter: "Individualisierbarkeit kann ein Mittel sein, die Barrieren für die Zugänglichkeit zu senken, indem man den besonderen Erfordernissen und/oder Fähigkeiten eines möglichst großen Benutzerkreisen [sic!] entgegenkommt."

Aus diesen Formulierungen ist erkennbar, dass Barrierefreiheit als ein Aspekt in den Bemühungen in Zusammenhang mit der Individualisierbarkeit einer Anwendung verstanden werden kann. Anders herum betrachtet, ist Individualisierbarkeit aber mehr als die Berücksichtigung von Barrierefreiheit. Deshalb wird das Ziel, weitgehend barrierefreie RIAs anbieten zu können, nun etwas breiter gefasst und im Kontext der Frage, wie RIAs ein Maximum an Individualisierbarkeit bieten können, betrachtet.

Die in Abschnitt 4.9 der ISO-Norm 9241-110 aufgezählten Anforderungen an die Individualisierbarkeit von Dialogsystemen können grob in drei Kategorien unterteilt werden:

1. Anforderungen an die visuelle Präsentation des Dialogs
2. Anforderungen an das Verhalten des Dialogs
3. Anforderungen an die inhaltliche Gestaltung des Dialogs

Wie Anforderungen aus der ersten Kategorie in RIAs berücksichtigt werden können, wurde bereits an früherer Stelle in dieser Arbeit erläutert (vgl. dazu Abschnitt 5.3.4 auf Seite 127). Ihre Umsetzung an das Verhalten des Dialogs liegt in der Verantwortung der Dialog-Entwickler. Die Anforderungen unterscheiden sich bei RIAs nicht von denen klassischer (Web- oder Desktop-)Anwendungen. Insofern werden sie nicht weiter in dieser Arbeit betrachtet. Wohl aber stellen die Anforderungen aus der dritten Kategorie (inhaltliche Gestaltung) in Zusammenhang mit RIAs neue Herausforderungen bzw. Chancen dar, auf die in dieser Arbeit eine Antwort gegeben werden soll.

6.1.2.1 Inhaltliche Individualisierbarkeit – Unterstützung mehrerer Sprachversionen

Die inhaltliche Gestaltung eines Dialogs (wobei hiermit fortan eine RIA gemeint sei) kann auf vielfältige Weise adressiert werden: so ist zum Beispiel die Unterstützung mehrerer Sprachen eine Form der Individualisierbarkeit der inhaltlichen Gestaltung. Fasst man den Sprachbegriff noch etwas weiter und sieht pro Sprache mehrere Varianten vor ("einfaches Deutsch", "gehobenes Deutsch", "lokal angepasstes Deutsch", "juristisch präzises Deutsch" etc.), so gelangt man zu Anforderung 4.9.4 der ISO-Norm 9241-110 ("Der Benutzer sollte [...] die Möglichkeit haben, eigenes Vokabular einzubinden [...]"). Mit Hilfe des MultiLanguageLabelers (siehe Abschnitt 5.3.3.2 auf Seite 123) steht bereits ein Werkzeug zur Verfügung, um RIAs sprachlich individualisierbar zu machen, sodass neben (Fremd-)Sprachen auch zielgruppengerechte Sprachversionen einer RIA problemlos angeboten werden können. Eine speziell für RIAs, die auch im Offline-Betrieb genutzt werden sollen, gedachte Erweiterung des MLL-Patterns wird in Abschnitt 6.5 betrachtet.

6.1.2.2 Inhaltliche Individualisierbarkeit – Anordnung der Bedienelemente

Als weitere Anforderung, die in den Kontext der inhaltlichen Individualisierbarkeit einzuordnen ist, nennt die Norm die Möglichkeit, "Dialogelemente oder Funktionen hinzuzufügen oder neu zu ordnen, insbesondere, um individuelle Bedürfnisse bei der Ausführung von Arbeitsaufgaben zu unterstützen" (Punkt 4.9.9). In früheren Arbeiten wurden dazu bereits Lösungsvorschläge in Form von *Design Pattern* erarbeitet (vgl. [Hei04a], [BBHM06a], [BBHM06b]). Zwar entstanden diese Arbeiten im Umfeld von Web-Portalen bzw. Plug-in-Systemen, jedoch sind sie hinreichend flexibel gehalten, um auch in RIAs Verwendung finden zu können. Insofern steht auch für diese Anforderung bereits eine Lösung zur Verfügung.

6.1.2.3 Inhaltliche Individualisierbarkeit – Anpassung an Rollen, Berechtigungen und Zustände

Der Wunsch, RIAs auch im Hinblick auf Berechtigungen des Benutzers zu individualisieren, ergibt sich nicht unmittelbar aus der oben genannten Norm. Er geht vielmehr auf die Besonderheiten zurück, die RIAs aufgrund ihres hybriden Charakters aufweisen, und führt zu offenen

Fragen, die – beginnend mit Abschnitt 6.2 – im restlichen Verlauf dieses Kapitels diskutiert und beantwortet werden sollen.

6.1.3 Weitere Begriffe: Personalisierbarkeit und *Customization*

Anstelle des gerade betrachteten Begriffs der inhaltlichen Individualisierbarkeit (bzw. Individualisierung) wird oftmals auch der Terminus *Personalisierbarkeit* (bzw. Personalisierung, englisch: *Personalization*) oder auch der Anglizismus *Customization* verwendet. Eine Abgrenzung dieser beiden Begriffe sowie des Begriffs der inhaltlichen Individualisierbarkeit ist nicht immer präzise möglich, insbesondere nicht bei einer Übertragung dieser Begriffe in die deutsche Sprache. In [Bla09] beschreibt der Autor Individualisierung als Weiterentwicklung von Personalisierung und bezieht sich dabei darauf, ob nur "bestimmte Aspekte" ("*certain aspects*") oder die gesamte Kommunikation auf eine Person zugeschnitten werden. Derart spitzfindige Unterscheidungsmerkmale sind für diese Arbeit jedoch nicht von Bedeutung.

6.2 RIAs, Web- und Desktop-Applikationen: Parallelen und Differenzen

Ein erklärtes, wenn nicht sogar *das* erklärte Ziel beim Einsatz von RIA-Technologien ist es, diese so zu konzipieren, dass sie sich dem Anwender so weit wie möglich als traditionelle Desktop-Anwendung und eben nicht als Web-Anwendung präsentieren. Weil sie *de facto* jedoch Web-Anwendungen sind, warten sie zugleich mit den Vorteilen dieses Genres auf. Das bedeutet, dass hier zwei zunächst völlig verschiedene Ansätze bei der Software-Entwicklung zusammengeführt werden müssen. Diese Unterschiede sollen zunächst einmal verdeutlicht werden.

6.2.1 Traditionelle Online-Anwendungen

Traditionelle Online-Anwendungen basieren auf dem klassischen Client-Server-Prinzip. Die Applikationslogik liegt ausschließlich auf dem Server, und dieser Server liefert dem Client einerseits die Eingabemöglichkeiten, die er zur Kommunikation mit der Applikation benötigt (typischerweise HTML-Formulare), sowie andererseits die Antwort auf die von ihm (dem Client) initiierten Anfragen. Der Client dient lediglich zur Visualisierung der Ausgabe sowie zum Entgegennehmen der Eingaben des Benutzers, die Verarbeitung der Eingaben sowie die Generierung der Ausgaben erfolgen jedoch ausschließlich auf der Serverseite. Somit kommt den Clients die Aufgabe eines Vermittlers zwischen Benutzer und Server(logik) zu. Clients dieser Art werden als *Thin Clients* bezeichnet. Durch den zusätzlichen Einsatz von JavaScript lässt sich ein solcher *Thin Client* zwar auch um eine "Verhaltensschicht" erweitern (d.h. Validierungen und ähnliches Feedback an den Benutzer können lokal vom Client behandelt werden), jedoch bleibt die grundsätzliche Trennung von lokalem Browser als Ein-/Ausgabemedium und dem Server, welcher die gesamte Logik der Anwendung enthält, auch dann bestehen.

6.2.1.1 Reaktionen auf die Benutzeridentifikation

Diese klassische Architektur von traditionellen Web-Applikationen erlaubt einen sehr hohen Grad an Personalisierbarkeit. Denn weil jede Ausgabe vom Server erst generiert wird (also

nicht in einer statischen Form bereits vorgehalten wird), kann der Server mit jeder neuen Antwort auf einen clientseitigen Kontakt eine individuell angepasste Antwort generieren. Die individuelle Anpassung der serverseitigen Antwort richtet sich dabei in der Regel nach dem momentanen Benutzer, der sich typischerweise zuvor durch die Eingabe seiner Zugangsdaten beim Server identifiziert hat.

Je nach Art der Web-Anwendung steht diese einem Benutzer erst nach erfolgter Anmeldung zur Verfügung. Andererseits ist es aber auch durchaus üblich, dass sich Web-Anwendungen in eingeschränktem Umfang auch ohne vorangegangene Anmeldung verwenden lassen und ihren Funktionsumfang nach erfolgreicher Anmeldung erweitern. Allein dies stellt bereits eine – noch recht profane – Reaktion auf die Benutzeridentifikation und mithin also eine *Individualisierung* dar.

Eine detailliertere Art, auf die Identifikation des Benutzers zu reagieren, als lediglich zwischen "ist angemeldet" und "ist nicht angemeldet" zu unterscheiden, ist ebenfalls geübte Praxis: verschiedene Benutzer können durchaus verschiedene Berechtigungen haben, was typischerweise in den Navigationsstrukturen (also im Funktionsmenü) zum Ausdruck kommt: diese passen sich nach erfolgreicher Anmeldung gleich den Berechtigungen des momentanen Benutzers an und zeigen demnach stets nur diejenigen Menüeinträge (bzw. sonstige Navigationselemente), die der Benutzer aufgrund der ihm zugeteilten Berechtigungen auch tatsächlich verwenden darf. Dies ist ebenfalls deshalb sehr einfach möglich, weil der Server auf jede Anfrage des Clients hin die auszugebende Seite zunächst generiert und somit jederzeit (d.h. mit jeder neuen Anfrage) auf Änderungen in den Berechtigungen reagieren kann.

Eine weitere Form der Reaktion auf die Benutzeridentifikation ist die inhaltliche Adaption der Ausgaben nach (bekannten oder vermuteten) Präferenzen des Benutzers. So kann beispielsweise jede Antwort des Servers auf eine Anfrage mit einigen Zusatzelementen angereichert werden, die speziell auf den momentanen Benutzer zugeschnitten sind. Ein etabliertes Beispiel für dieses Vorgehen ist bei Online-Buchhändlern zu beobachten: ein Teil der Antwort auf clientseitige Anfragen enthält auch stets Empfehlungen von Büchern oder andere Medien, die für den Benutzer von Interesse sein könnten.

6.2.1.2 Reaktion auf Änderungen in der Konnektivität

Ihrer Natur nach sind Web-Anwendungen nur zu verwenden, wenn eine Konnektivität zwischen Client und Server (d.h. eine Online-Verbindung) besteht. Im seitenzentrierten Ansatz klassischer Web-Anwendungen würde ein Verlust der Konnektivität (erst) beim nächten Kontaktversuch mit dem Server auffallen. Dann nämlich schlägt dieser Versuch fehl. Da der Server aufgrund der abgebrochenen Verbindung nun nicht mehr mit einer entsprechenden Meldung reagieren kann, sind es der verwendete Browser oder das Betriebssystem, die stattdessen einen Hinweis auf das Konnektivitätsproblem geben.

Weil die serverseitige Applikation vom Abbruch der Verbindung keine Kenntnis erlangt, verbleibt sie – in der Regel – für ein definiertes Zeitfenster (Frist) in dem den jeweiligen Benutzer betreffenden letzten Zustand und setzt diesen nach Ablauf des Zeitfensters so zurück, als sei der Benutzer nicht angemeldet ("*time out*"). Dieser Mechanismus hilft, einen Missbrauch von nicht ordentlich beendeten Sitzungen zu verhindern, und greift oftmals nicht nur beim Verlust der Konnektivität, sondern auch bei einer Inaktivität des Benutzers über die vorgegebene Frist hinaus.

Die Erwähnung dieses durchaus etablierten Vorgehens ist deshalb interessant, weil es abermals eine rein serverseitige Reaktion demonstriert. Die Applikation auf dem Server verwaltet die

Kommunikation mit den angemeldeten Benutzern und reagiert nach einer (zeitlich definierten) Phase der Inaktivität beispielsweise mit einem *time out* nebst automatischer Abmeldung des betreffenden Benutzers. Die Reaktion kann indes nicht durch den Server an den Client kommuniziert werden: dazu müsste zunächst ein erneuter Kontaktversuch des Clients abgewartet werden, doch dieser kann wegen der nicht mehr vorhandenen Konnektivität nicht mehr erfolgen.

Ein ähnliches, wie das zuvor beschriebene Problem des Verfolgens des Konnektivitätsstatus' eines (bzw. aller) Benutzer(s), ergibt sich auch aus Änderungen im Login-Status, also jenes Zustands, der angibt, ob ein bestimmter Benutzer gerade angemeldet ist oder nicht. Typischerweise sind Funktionen, die erst nach einer erfolgreichen Anmeldung zur Verfügung stehen, nicht in den Menü- und Navigationsstrukturen enthalten, solange diese Anmeldung nicht erfolgte. Damit sind sie also auf "normalem Wege" nicht erreichbar. Es wäre jedoch ein fataler Irrtum, wenn einzig aus der Tatsache, dass eine Funktion nicht im Menü erwähnt wird, geschlossen würde, dass sie dann auch nicht aufgerufen werden kann. Denn diese vermeintliche "Sicherheitsvorkehrung" liesse sich bereits durch eine simple Manipulation der Anfrageparameter aushebeln. Dann stünde eine solche Funktion eben auch ohne vorherige Anmeldung zur Verfügung. Aus diesem Grund sollte der Server, bevor er eine entsprechend berechtigungsrelevante Funktion ausführt, sichergestellt haben, dass eine Anmeldung erfolgte und der Benutzer über die erforderlichen Berechtigungen verfügt.

6.2.1.3 Benutzerpräferenzen

Weil traditionelle Web-Anwendungen nahezu ausschließlich serverseitig gesteuert werden, müssen auch dort – also serverseitig – die Benutzereinstellungen verwaltet und gespeichert werden (sofern dies im Funktionsangebot der Anwendung vorgesehen ist). Exemplarisch seien hier zwei solcher möglichen benutzerspezifischen Einstellungen genannt.

Spracheinstellungen: Der Benutzer trifft seine Wahl der Sprache, und fortan werden die zur Anzeige zu bringenden Seiten stets in dieser Sprache generiert. Eine Änderung der Spracheinstellung wird mithin also mit dem nächsten Aufruf einer Seite wirksam.

Anordnungseinstellungen: Gerade bei umfangreichen Web-Anwendungen wie Portalen ist die Frage von Bedeutung, wie ein Benutzer eine ihm angenehme Auswahl, Anordnung und Darstellung von Informationen für sich arrangieren und vor allem als seine persönliche Präferenz derart persistieren kann, dass sie bei jeder neuen Anmeldung wieder zur Anwendung kommt. Aus der Tatsache, dass (bei Portalen) die zur Anzeige zu bringenden Seiten stets generiert werden, folgt, dass demnach auch serverseitig die Benutzerpräferenzen gespeichert werden müssen. Ist dies der Fall, so kann sich jeder Aufbau einer Seite an den Einstellungen des Benutzers orientieren (siehe hierzu beispielsweise [Hei04a], aber auch [BBHM06a] und [BBHM06b]).

Layouteinstellungen: Auch der bereits mehrfach genannte Umstand, dass manche Benutzer aufgrund ihres eingeschränkten Sehvermögens auf besonders große Schriften oder eine kontraststarke – ggf. auch monochrome – Darstellung angewiesen sind, ist in diesem Zusammenhang zu nennen. Um ihm gerecht zu werden, kann der Server verschiedene Stylesheets bereit halten und das jeweils zum aktuellen Benutzer passende ausliefern.

6.2.1.4 Bezug zu RIAs

Die in diesem Abschnitt bislang aufgezeigten Frage- und Problemstellungen bei traditionellen Web-Anwendungen sind bereits auf vielfältige Weise gelöst worden und stellen heutzutage keine Herausforderung mehr dar.

Anders verhält es sich jedoch, wenn man die sich aus den geschilderten Problemen ergebenden Fragestellungen in der Welt der RIAs beantworten möchte:

- Wie kann und soll eine RIA auf Änderungen im Konnektivitätsstatus reagieren?
- Wie geht eine RIA mit Änderungen im Login-Status um?
- Wie kann eine RIA ihre Navigationsstrukturen (funktional) an den momentanen Benutzer anpassen?
- Wie lassen sich möglichst flexibel unterschiedliche Sprachversionen realisieren?
- Wie kann eine RIA ihr Layout zur Laufzeit aufgrund von Benutzereinstellungen variieren?

Die Beantwortung dieser Fragen im RIA-Kontext unterscheidet sich deshalb fundamental von jenen im traditionellen Web-Anwendungskontext, weil RIAs ein in sich geschlossenes Softwaresystem darstellen, das auf dem Client-Rechner läuft. Analog zu den Erläuterungen in Zusammenhang mit traditionellen Web-Anwendungen ist es auch bei RIAs nicht ungewöhnlich, dass sie, solange noch keine Benutzeranmeldung erfolgte, ihren Funktionsumfang einschränken und erst nach erfolgreicher Authentifizierung alle Funktionen freigeben. Ebenso muss die Möglichkeit existieren, dass verschiedene Benutzer – je nach Berechtigungen – verschiedene Funktionen zur Verfügung stehen haben. Und auch der Wunsch, verschiedene Sprach- und/oder Layoutversionen einer RIA anzubieten, ist von gleichwertiger Bedeutung wie bei traditionellen Web-Anwendungen. Zu den letztgenannten Punkten wurden bereits im vorangegangenen Kapitel Lösungsansätze vorgestellt (siehe Abschnitt 5.3.3 auf Seite 122 sowie Abschnitt 5.5.4.1 auf Seite 146). Sie sollen hier eine Wiederverwendung erfahren.

Die Herangehensweise an die übrigen der oben genannten Frage- und Problemstellungen unterscheidet sich indes in Zusammenhang mit RIAs sehr stark von derjenigen bei traditionellen Web-Anwendungen. Dies hat seine Ursache darin, dass RIAs eine in sich geschlossene Anwendung darstellen und nicht serverseitig mit darzustellendem Inhalt beliefert werden. Dies bedeutet in der Konsequenz, dass RIAs demnach auf jedem Client-Rechner bereits einen vollständigen Funktionsumfang in sich bergen, gleichwohl aber nur diejenige Teilmenge ihres Funktionsumfangs in den Navigationsstrukturen bzw. Menüs angeboten werden darf, die den Berechtigungen oder dem Login-Status des momentanen Benutzers entspricht. Ein bedeutender Teil der "Verantwortung", welcher bislang dem Server oblag, wird auf diese Weise in den Client verlagert.

Auch bei der Frage nach der Unterstützung mehrerer Sprachen ist bei RIAs ein fundamentaler Unterschied zu traditionellen Web-Anwendungen erkennbar. Die angezeigten Inhalte werden nicht serverseitig in der gewünschten Sprache generiert, sondern clientseitig verarbeitet. Mithin muss die gesamte RIA für die Unterstützung der Mehrsprachigkeit konzipiert sein. Keine (empfehlenswerte) Lösung wäre das Vorhalten je einer Version der RIA pro Sprache. Ebensowenig wäre das Vorhalten einer RIA zu empfehlen, die *alle* Sprachen unterstützt, denn dies würde ihren Umfang entsprechend vergrößern. Web-Inhalte – gleich welcher Art – sollten jedoch stets

so wenig Datenvolumen wie möglich besitzen, denn je kleiner die zu ladende Dateigröße ist, desto schneller steht der Inhalt zur Verfügung. Als Lösung für dieses Problem bietet sich an, die Tatsache zu nutzen, dass RIAs online verwendet werden, denn so können sie bei Bedarf eine gewünschte Sprachdefinition nachladen. Eine geeignete Lösung lässt sich ebenfalls für den Fall finden, dass eine RIA auch im offline-Betrieb genutzt werden soll. Diese wird in Abschnitt 6.5 vorgestellt.

6.2.2 Zusammenfassende Problembeschreibung

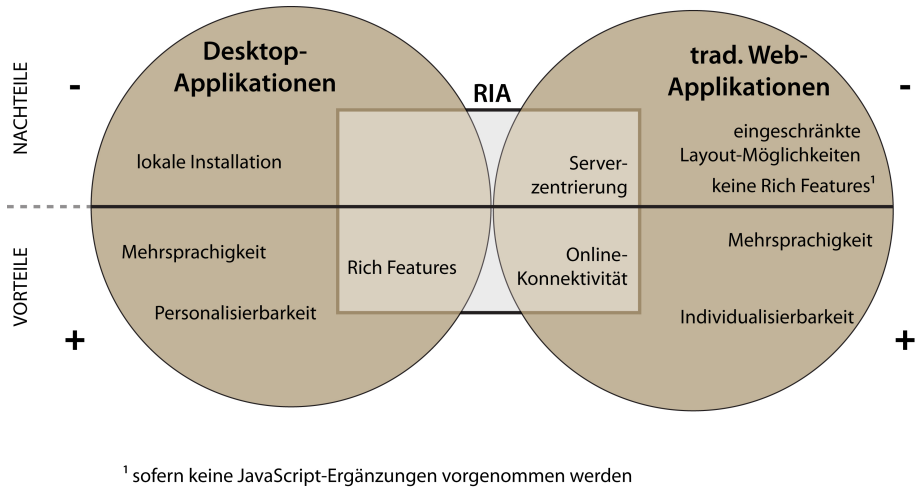


Abbildung 6.1: Desktop- vs. traditionelle Web-Applikationen: Unterschiede/Gemeinsamkeiten

Einen zusammenfassenden Überblick über die Unterschiede und Gemeinsamkeiten zwischen RIAs und traditionellen Web-Applikationen gibt Abbildung 6.1. Sie zeigt die Vor- und Nachteile von Desktop- bzw. Web-Applikationen. Der helle Bereich in der Mitte skizziert, welche Fähigkeiten von Desktop- und Web-Applikationen in RIAs vereint werden (Schnittmenge). Aus diesem Bereich ist ersichtlich, dass RIAs noch nicht alle Vorteile der jeweiligen Applikationstypen nativ unterstützen. Anhand Abbildung 6.1 können die in diesem Kapitel beabsichtigten Ziele wie folgt veranschaulicht werden:

- Für die als positiv bewerteten und noch nicht oder nur in Ansätzen in RIAs enthaltenen *Features* von Web- und Desktop-Applikationen sollen allgemein zu verwendende Entwurfsmuster gefunden werden, mit Hilfe derer RIAs um diese Features erweitert werden können.

- Die als negativ bewerteten Merkmale sollen dabei *nicht* tangiert werden.

- Sofern die Merkmale aus der Schnittmenge nicht unmittelbar aus dem Bereich der Web- bzw. Desktop-Applikationen auf RIAs übertragen werden können, sollen die in diesem Kapitel vorgestellten Entwurfsmuster auch dafür Lösungsmöglichkeit aufzeigen.

Die Diskussion der Übertragung der in diesem Kapitel vorgestellten Entwurfsmuster auf ein konkretes, praxisnahes Softwareprojekt schließt sich in Kapitel 7 an.

6.3 Status Sensitive Components

In diesem Abschnitt erfolgt die Präsentation eines Lösungsansatzes für den Umgang mit Änderungen bestimmter Zustände einer RIA zur Laufzeit und legt damit einen Grundstock für die Möglichkeit, RIAs (zur Laufzeit) zu individualisieren. Die Zustände, deren Änderungen hier interessieren, können beispielsweise sein:

- der Login-Status des Benutzers (Ist ein Benutzer angemeldet oder nicht?),
- der aktuelle Konnektivitäts-Status (Ist die Applikation online oder nicht?) oder
- der Berechtigungs-Status eines angemeldeten Benutzers (Über welche Berechtigungen/ Privilegien verfügt er?)

Diese drei Zustände sollen dabei nur als Beispiele für mögliche Einsatzszenarien verstanden werden, denn der präsentierte Lösungsansatz ist frei skalierbar und erlaubt somit, auch beliebige andere "interessierende" Zustände zu überwachen.

Zur Laufzeit können sich die genannten Zustände jederzeit ändern. Während dies bei herkömmlichen Web-Applikationen unproblematisch ist (der Server kann stets zustandsadäquat antworten), müssen RIAs ihre Präsentation dynamisch an die einzelnen Zustände oder gar an Kombinationen von Zuständen anpassen können. Andernfalls könnten Funktionen zur Verfügung stehen, die aufgrund einer gerade eingetretenen Zustandsänderung gar nicht mehr zur Verfügung stehen dürften.

Unter dem Titel "*Status Sensitive Components: Adapting Rich Internet Applications to their Runtime Context*" wurde in [HS09b] für dieses Problem ein Lösungsansatz in Form eines Design-Patterns präsentiert, wobei dort einige Ideen aus Gründen der Beschränkung der Seitenzahl nur sehr knapp erläutert werden konnten. In diesem Abschnitt werden die Idee sowie das daraus resultierende Pattern (im Folgenden "SSC-Pattern" genannt) detaillierter dargestellt.

Zunächst sollen zwei Begriffe, die im Rahmen der nachstehenden Ausführungen immer wieder verwendet werden, eine Definition erfahren, um deren Bedeutung im Rahmen dieser Arbeit zu klären:

Definition: Zustand (engl.: *Status*) Der Begriff "Zustand" bzw. sein im weiteren Verlauf verwendetes englischsprachiges Pendant "*Status*" einer RIA bezeichnet die Ausprägung einer bestimmten Eigenschaft einer RIA zur Laufzeit. Wenn diese Eigenschaft überwacht wird, sodass auf ihre Änderungen reagiert werden kann, wird der Begriff "überwachter Zustand" ("*Watched Status*") verwendet.

Das SSC-Pattern erlaubt die zeitgleiche Überwachung beliebig vieler Zustände. Ziel des Patterns ist es, dass Änderungen verschiedener überwachter Zustände der RIA erkannt und behandelt werden. Das Bestehen bzw. Nicht-Bestehen einer Online-Verbindung ist ein Beispiel für einen überwachbaren Status ("*Online Status*"), ein weiteres ist, ob der Benutzer angemeldet ist oder nicht ("*Login Status*"). Ein drittes Beispiel ist der Privilegien-Status, der angibt, über welche Rechte der aktuelle Benutzer verfügt. Über diese Beispiele hinaus ist das Pattern aber sehr leicht auf beliebig viele weitere zu überwachende Zustände erweiterbar.

Aus der Betrachtung der Menge aller überwachten Zustände (oder auch einer beliebigen Teilmenge davon) zur Laufzeit geht ein weiterer Begriff hervor:

Definition: Kontext (engl.: *Context*) Die Gesamtheit der Eigenschaften aller überwachten Zustände einer RIA zu einem beliebigen Moment während der Laufzeit wird als "(aktueller) Kontext" (englisch: *[Actual] Context*) bezeichnet.

Mit Hilfe der beiden soeben definierten Begriffe kann das Problem, das in diesem Abschnitt eine Lösungsmöglichkeit erfahren soll, wie folgt ausgedrückt werden:

> *Wie kann die Präsentationsschicht der RIA flexibel auf Änderungen ihres Kontexts reagieren?*

Die nachstehenden Beispiele mögen dieses Problem verdeutlichen:

Beispiel 1: Wenn die RIA offline ist, d.h. wenn die überwachte Eigenschaft, ob eine Online-Verbindung besteht oder nicht ("Konnektivitäts-Status"), die Ausprägung "offline" besitzt, sollen diejenigen Funktionen, die das Bestehen einer Online-Verbindung voraussetzen, deaktiviert oder ganz aus dem momentan verwendbaren Funktionskanon entfernt werden.

Beispiel 2: Funktionen, die eine Anmeldung des Benutzers voraussetzen, dürfen nur dann zugänglich sein, wenn zuvor ein Anmeldevorgang erfolgreich stattfand.

Dieses Beispiel muss in zwei Richtungen gelten: zum einen muss die Menge der angebotenen Funktionen erweitert werden, wenn die Anmeldung erfolgte, zum anderen müssen diese Funktionen aber wieder entfernt werden, wenn sich der Zustand geändert, d.h. der Benutzer die Abmeldung veranlasst hat.

Während es für den Zugang zu manchen Funktionen bereits ausreichend sein kann, dass sich ein Benutzer lediglich angemeldet hat, ist für andere zudem noch von Bedeutung, *wer* sich angemeldet hat, d.h über welche Berechtigungen und Privilegien der aktuelle Benutzer verfügt. Dies führt zu

Beispiel 3: Je nach den Berechtigungen, die einem momentan angemeldeten Benutzer erteilt wurden, stehen bestimmte Funktionen nicht oder gerade *doch* zur Verfügung. Auch hier muss sowohl das Hinzunehmen als auch das Entfernen von Funktionen veranlasst werden.

Wie in der Einführung dieses Kapitels erläutert, werden die in den Beispielen angedeuteten Problemstellungen in traditionellen Web-Applikationen dadurch gelöst, dass der Server typischerweise auf jede Anfrage des Clients hin seine Antwort(seite) neu generiert. Dabei werden die Zustände serverseitig verwaltet und berücksichtigt, und dementsprechend fällt die Antwort an den Client stets passend aus. Eine RIA steht jedoch in vollem Funktionsumfang auf dem Client-Rechner zur Verfügung, sodass dafür Sorge getragen werden muss, dass dort – also in der RIA – nur diejenige Teilmenge des insgesamt verfügbaren Funktionsumfangs dargestellt wird, die adäquat zum aktuellen Kontext ist. Die RIA muss also "sensibel" für die von ihr überwachten Zustände, d.h. – um den für dieses Pattern namensgebenden Anglizismus zu verwenden – "*status sensitive*" sein. (Dies erklärt die ersten beiden Wortbestandteile des Namens des hier präsentierten Patterns.) Der dritte Begriff in der Namensgebung des Patterns nennt das Wort *components* (Komponenten). Damit soll zum Ausdruck gebracht werden, dass es nicht die RIA als solches ist, die auf Kontextänderungen reagiert, sondern (einige) ihre(r) Bestandteile (Komponenten).

6.3.1 Zustandssensitive Komponenten

Unter dem gerade verwendeten Begriff "Bestandteile (Komponenten) einer RIA" sind die verschiedenen Elemente einer grafischen Benutzeroberfläche (GUI) zu verstehen, wie z.B. Menüstrukturen, Schaltflächen oder auch komplette Container, die jeweils eine Menge weiterer derartiger Elemente verwalten.

Bei Verwendung einer objektorientierten Programmiersprache stellt jede Komponente ein eigenes Objekt dar, d.h. es ist eine Instanz einer entsprechenden Klasse. Das SSC-Pattern sieht vor, dass die Komponentenklassen um die Implementierung "gewisser" Interfaces erweitert werden. Jedes dieser Interfaces steht dabei für genau einen zu überwachenden Zustand. Aus der Benennung eines Interfaces lässt sich dann unmittelbar erkennen, welcher Zustand durch dieses Interface überwacht wird. Weil eine Klasse (hier: Komponente) beliebig viele Interfaces implementieren kann, ist sie demnach auch in der Lage, eine beliebige Kombination von Zuständen zu überwachen. Gleichzeitig ermöglicht das SSC-Pattern aber auch die individuelle "Sensibilisierung"[1] jeder einzelnen Komponente unabhängig von allen anderen Komponenten.

Wenn sich der Kontext ändert, d.h. wenn also einer der überwachten Zustände eine Änderung erfährt, so müssen die entsprechend sensitiven Komponenten benachrichtigt werden. Grundsätzlich kommen hierfür zwei Mechanismen in Frage:

- Ereignisbehandlung (*Event-Handling*)
- Messaging bzw. Publish-Subscribe

Beide Mechanismen beruhen letztendlich auf dem Observer Pattern (vgl. dazu beispielsweise [GHJV94]). Die genaue Unterscheidung zwischen ihnen kann z.B. in [HW04] nachgelesen werden, sie ist für das Verständnis des SSC-Patterns jedoch nicht von Bedeutung, denn das SSC-Pattern abstrahiert den verwendeten Benachrichtigungsmechanismus und bleibt somit unabhängig von diesem. Aus Sicht der Komponenten stellt es sich als *Black Box* dar, an der sich die Komponenten für den Erhalt von Benachrichtigungen an- oder abmelden, ohne Kenntnis der zugrundeliegenden Funktionsweise zu besitzen. Wie in [HW04] beschrieben wird, basiert das Event-Handling oftmals auf einem Publish-Subscribe-Mechanismus. Letzterer gilt als robust und etabliert, sodass sowohl für diesen als auch für das auf ihm beruhende Event-Handling – und damit auch für das SSC-Pattern – gilt, dass sie "sehr gut skalierbar [sind]" und "das Potenzial für eine einfache Integration von autonomen, heterogenen Komponenten in komplexe Systeme, die sich einfach entwickeln und skalieren lassen, [enthalten]" (eigene Übersetzung aus [MFP06]).

6.3.2 Das SSC-Pattern

Nach der einführenden Begriffsbestimmung und der verbalen Übersicht im vorangegangenen Abschnitt soll das SSC-Pattern nun anhand zweier UML-Diagramme erläutert werden.

Abbildung 6.2 zeigt ein UML-Klassendiagramm des SSC-Patterns. Die beteiligten Klassen bzw. Interfaces haben dabei die folgenden Bedeutungen:

[1] Mit dieser Bezeichnung sei die Tatsache gemeint, dass eine Komponente auf Änderungen des Kontexts reagiert.

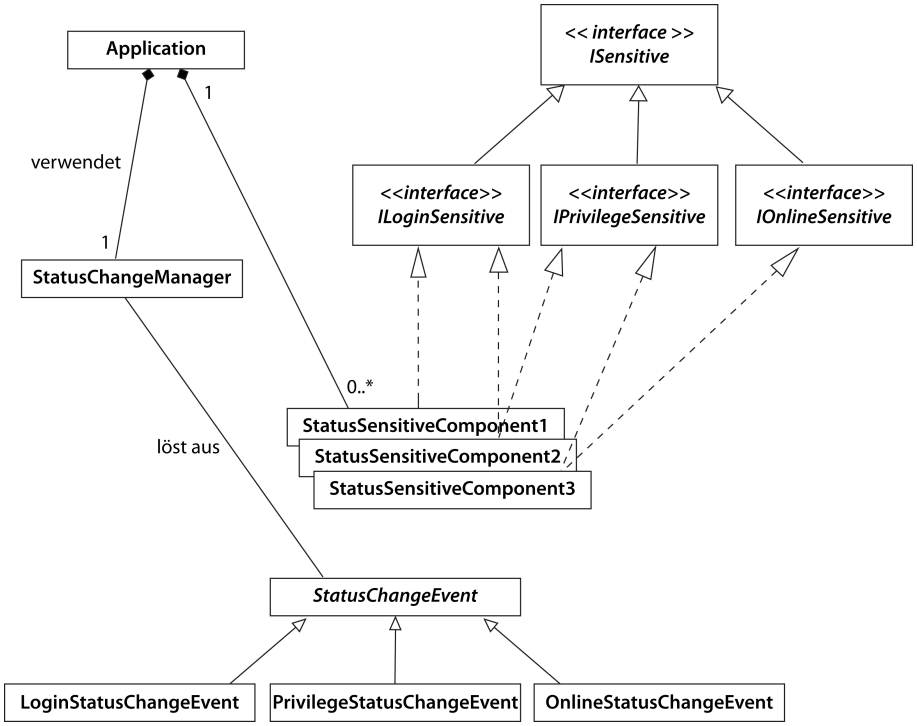


Abbildung 6.2: UML-Klassendiagramm des SSC-Patterns

6.3.2.1 Die Klasse `Application`

Die Klasse `Application` (im Folgenden in deutsch als "Applikation" bezeichnet) repräsentiert die eigentliche RIA, enthält also die Anwendungslogik und kontrolliert bzw. steuert den gesamten Programmablauf. Insbesondere stellt diese Klasse auch den Einstiegspunkt in die Anwendung bereit. Eine Applikation kann eine beliebige Menge bzw. Kombination von sensitiven Komponenten verwenden, besitzt aber stets genau einen *StatusChangeManager* (vgl. Abschnitt 6.3.4 auf Seite 166). Wenn sich einer der überwachten Zustände zur Laufzeit ändert, benachrichtigt die Applikation mit Hilfe des *StatusChangeManager* alle diejenigen Komponenten, die als sensitiv für den veränderten Zustand registriert sind. Die Reaktion auf die jeweilige Änderung obliegt dann individuell jeder einzelnen der benachrichtigten Komponenten.

6.3.2.2 Die Hierarchie der `ISensitive`-Interfaces

Das Herzstück des SSC-Patterns stellen die Erweiterungen des Interfaces `ISensitive` dar. Jede dieser Erweiterungen repräsentiert genau einen überwachten Zustand und deklariert diejenige Funktion, die aufgerufen wird, wenn sich der überwachte Zustand ändert. Das UML-Sequenzdiagramm im nachfolgenden Abschnitt (Abbildung 6.3 auf Seite 168) visualisiert diesen Prozess.

Komponenten, die auf die Änderung eines oder mehrerer Zustände reagieren sollen, müssen nun lediglich die entsprechenden Interfaces implementieren und sind somit fortan "sensibilisert", d.h. *status sensitive*.

Listing 6.1 zeigt das Interface `ISensitive`. Die Methode `onStatusChange` ist diejenige Methode, die aufgerufen wird, wenn sich der überwachte Zustand ändert. Die anderen beiden

Methoden dienen dazu, dass sich die Komponenten, die dieses Interface implementieren, beim `StatusChangeManager` an- und abmelden können.

Listing 6.1: Das Interface `ISensitive`

```
1 public interface ISensitive {
2   function register() : void;
3   function unregister() : void;
4   function onStatusChange (ev: StatusChangeEvent) : void;
5 }
```

Die drei in Abbildung 6.2 gezeigten Erweiterungen dieses Interfaces (`ILoginSensitive`, `IOnlineSensitive` und `IPrivilegeSensitive`) stehen beispielhaft für drei verschiedene überwachte Zustände und werden auch im Quellcode lediglich als simple Erweiterung des Interfaces `ISensitive` definiert, ohne dies notwendigerweise zu erweitern:

```
public interface ILoginSensitive extends ISensitive {}
```

Sie sollen hier nur als Beispiele verstanden werden, denn das SSC-Pattern erlaubt grundsätzlich die Verwendung beliebig vieler Ableitungen des Interfaces `ISensitive`.

6.3.3 Die Hierarchie der `StatusSensitiveComponents`

Die im Klassendiagramm mit der Bezeichnung `StatusSensitiveComponent` (1-3) benannten Klassen sind die konkreten GUI-Komponenten, die auf Zustandsänderungen reagieren sollen. Jede dieser Klasse kann – unabhängig von allen anderen – eine beliebige Kombination von `ISensitive`-Interfaces (bzw. dessen Ableitungen) implementieren und auf diese Weise zum Ausdruck bringen, welchen Zustand (oder: welche Zustände) sie überwachen will. Weil sich jede Kombination von Interfaces durch die Komponenten implementieren lässt, kann auch jede gewünschte Sensitivität erreicht werden.

In Listing 6.2 wird die Implementierung einer *StatusSensitiveComponent* am Beispiel der Klasse `Canvas` demonstriert. Die Klasse `Canvas` steht hierbei für einen Container, der weitere GUI-Elemente aufnehmen kann. Der Komponente werden im Beispiel zwei Sensitivitäten hinzugefügt: somit überwacht sie einerseits den Login-Status und andererseits auch die Konnektivität (d.h. den Online-Status).

Wenn zur Laufzeit eine Änderung in einem der überwachten Zustände (oder auch in mehreren) auftritt, werden durch die Applikation alle entsprechend sensitiven Komponenten benachrichtigt, indem deren Methode (`onStatusChange`) aufgerufen wird. Dieses Vorgehen erlaubt, dass jede Komponente autark (d.h. unabhängig von allen anderen Komponenten auf die Zustandsänderung reagieren) kann. Unabhängig davon kann natürlich auch die Applikation selber sensitiv sein und ebenfalls auf das eingetretene Ereignis reagieren.

Der Methode `onStatusChange` wird ein Parameter mitgegeben, über den zur Laufzeit nähere Informationen über die Zustandsänderung bezogen werden können. Im Quelltext ist dieser Parameter unter der Bezeichnung `scev` als `StatusChangeEvent` typisiert. Diese Klasse ist jedoch abstrakt (vgl. Abbildung 6.2), sodass dieser Parameter zur Laufzeit die Instanz einer konkreten Klasse sein muss, d.h. hier: eine Instanz einer von `StatusChangeEvent` abgeleiteten Klasse. Bei jedem Aufruf der Methode `onStatusChange` kann die Komponente also Kenntnis darüber erlangen, *welcher* Zustand sich aktuell verändert hat, indem der Laufzeit-Typ des Parameters `scev` ermittelt wird.

Wenngleich nach außen hin lediglich die Methode `onStatusChange` zur Verfügung steht, erlaubt die Abfrage des Laufzeit-Typs des Parameters `scev`, auf verschiedene Zustandsänderungen auch verschieden zu reagieren. Dies ist jedoch eine individuelle Eigenschaft einer jeden Komponente und lässt sich durch die einzige von außen erreichbare Methode `onStatusChange` sauber kapseln.

Listing 6.2: Beispiel einer Klasse die das Interface ISensitive implementiert

```
1  public class LoginAndOnlineSensitiveCanvas
2    extends Canvas
3    implements ILoginSensitive, IOnlineSensitive
4  {
5    public function LoginSensitiveCanvas()
6    {
7      super();
8      register();
9    }
10
11   public function register():void
12   {
13     StatusChangeManager.subscribe(this);
14   }
15
16   public function unregister():void
17   {
18     StatusChangeManager.unsubscribe(this);
19   }
20
21   public function onStatusChange
22     (scev:StatusChangeEvent):void
23   {
24     // react depending of concrete status type
25   }
26 }
```

6.3.4 Die Klasse `StatusChangeManager`

Die Klasse `StatusChangeManager` verwaltet die Registrierungen und Benachrichtigungen aller zustandssensitiven Komponenten. Wie zuvor bereits erwähnt, besitzt jede Applikation genau einen `StatusChangeManager`. Deshalb kann dieser wahlweise aus rein statischen Methoden (Klassenmethoden) bestehen oder – sofern er ebenfalls als Objekt instanziiert werden soll – sich unter Verwendung des Singleton Patterns (vgl. [GHJV94]) genau einmal instanziieren lassen.

Ein `StatusChangeManager` wartet mit zwei öffentlichen Methoden auf, mit denen sich die Komponenten registrieren (anmelden) und wieder abmelden können:

Listing 6.3: Methoden zum An- und Abmelden beim `StatusChangeManager`

```
1  // subscribe given component for notification
2 public static function subscribe (lw: ISensitive) : void {
3   // ... [implementation omitted] ...
4 }
5
6 // unsubscribe given component from notification
7 public static function unsubscribe (lw: ISensitive) : void {
8   // ... [implementation omitted] ...
9 }
```

Die eingangs erwähnte Unabhängigkeit des Patterns vom Mechanismus zur Benachrichtigung der Komponenten kommt an dieser Stelle zum Ausdruck: der `StatusChangeManager` kapselt den zugrunde liegenden Mechanismus dergestalt, dass die Komponenten lediglich die zuvor genannten zwei Methoden verwenden, um sich für eine Benachrichtigung an- oder abzumelden. Die übrige Implementierung des `StatusChangeManager` bleibt nach aussen hin verborgen, sodass in Programmiersprachen, die das *Event Handling* unterstützen, hiervon Gebrauch gemacht werden kann. Andernfalls lässt sich aber auch eine herkömmliche Publish-/Subscribe-Architektur oder jeder andere Benachrichtigungsmechanismus verwenden, ohne dass dies eine Änderung des Quellcodes an anderer Stelle als im `StatusChangeManager` erforderlich macht, denn nur dort wird der Benachrichtigungsmechanismus konkret festgelegt.

6.3.5 Die `StatusChangeEvent`-Hierarchie

Im Abschnitt über `StatusSensitiveComponents` wurde bereits die Bedeutung der Klassenhierarchie der `StatusChangeEvents` kurz angedeutet: in ihr enthaltene Klassen repräsentieren jeweils genau ein spezielles eingetretenes Ereignis, das an die "interessierten" Komponenten (ggf. angereichert durch präzisere Informationen zum neuen Zustand) kommuniziert wird. Die Klassen dieser Hierarchie agieren damit wie `Event`-Objekte einer ereignisbasierten Programmiersprache, müssen aber nicht unbedingt ein solches sein. Mit anderen Worten: der zugrundeliegende Benachrichtigungsmechanismus wird durch die StatusChangeEvent-Hierarchie gekapselt: er *kann* durch Event-Basierung implementiert werden, muss dies aber nicht.

Die Oberklasse der `StatusChangeEvent-Hierarchie` ist abstrakt und kann deshalb zur Laufzeit nicht verwendet – d.h. instanziiert – werden. Vielmehr sollen zur Laufzeit ihre konkreten Unterklassen Verwendung finden. Die Idee des SSC-Patterns an dieser Stelle ist es jedoch *nicht*, dass die Unterklassen der `StatusChangeEvent-Hierarchie` um weitere Felder oder Methoden angereichert werden, sondern sie sollen als Repräsentierung der eingetretenen Zustandsänderung dienen. Durch eine Abfrage des Laufzeit-Typs für diese Komponenten soll also erkennbar sein, welche Zustandsänderung aktuell eingetreten ist.

6.3.6 Das SSC-Pattern als UML-Sequenz-Diagramm

Nach der Vorstellung aller am SSC-Pattern beteiligten Akteure (d.h. Klassen und Interfaces) soll abschließend die Funktionsweise des SSC-Patterns mit Hilfe des UML-Sequenzdiagramms in Abbildung 6.3 visualisiert werden.

Zu Beginn instanziiert die Applikation genau einen *StatusChangeManager* sowie alle benötigten (zustandssensitiven) Komponenten. Jede Komponente registriert sich durch die Methode `subscribe` beim `statusChangeManager`. Wenn zur Laufzeit eine Änderung eines Zustands eintritt (im Diagramm durch das `dispatchStatusChangeEvent` zum Ausdruck gebracht), führt dies zum Aufruf der Methode `onStatusChange` für jede der registrierten Komponenten. Dies ist derjenige Teil des Ablaufs, der wahlweise durch Event-Handling oder durch andere Messaging-Mechanismen realisiert werden kann. Jede Komponente erhält durch den Parameter `scev` der Methode `onStatusChange` Kenntnis über die aktuell eingetretene Zustandsänderung und kann individuell darauf reagieren.

Wenn zur Laufzeit der Applikation neue GUI-Container zur Anzeige gebracht werden sollen, die bei Programmstart noch nicht instanziiert waren, so wird der in Abbildung 6.3 gezeigte Ablauf auf diesen Container angewendet, d.h. dessen Komponenten durchlaufen die Registrierungsschritte, um bei der nächsten Zustandsänderung entsprechend informiert zu werden.

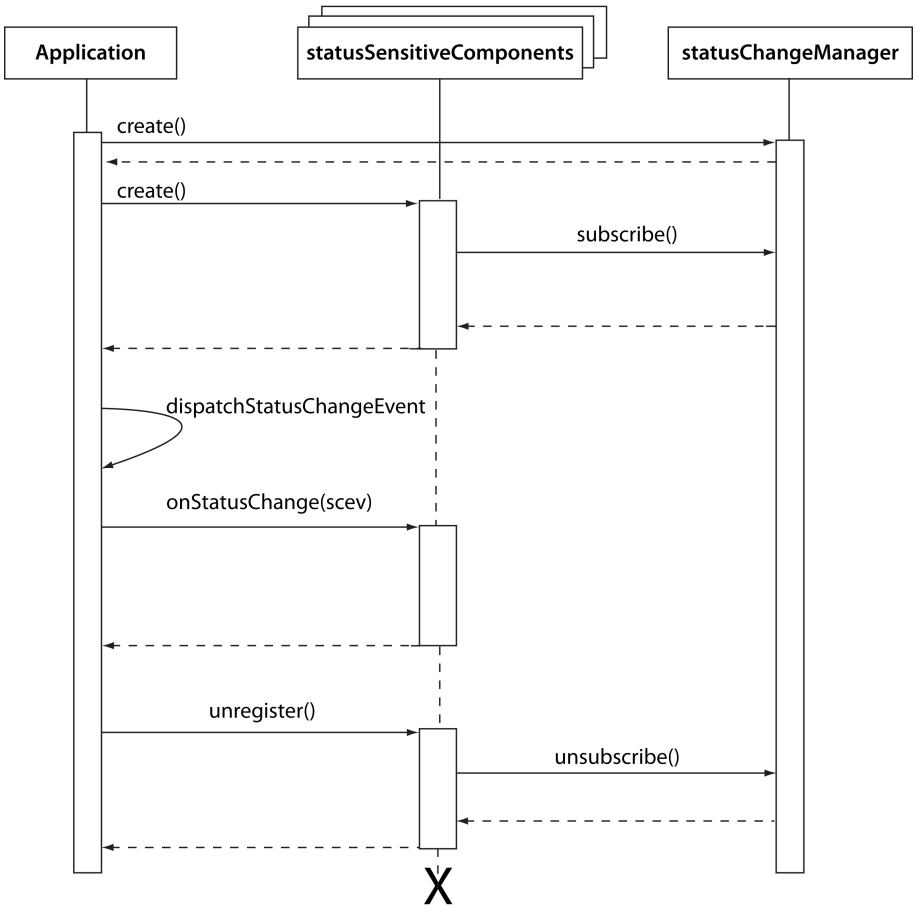


Abbildung 6.3: UML-Sequenzdiagramm zum SSC-Pattern

Um keine unerwünschten Seiteneffekte zu bekommen, ist es notwendig, dass sich die Komponenten beim *StatusChangeManager* auch wieder abmelden, wenn der GUI-Container nicht mehr angezeigt wird. Dies entspricht den unteren beiden Schritten im Sequenzdiagramm: durch den Aufruf ihrer jeweiligen Methoden `unregister` veranlasst die Applikation die Komponenten, sich von den Benachrichtigungen des *StatusChangeManagers* ausnehmen zu lassen. Die Komponenten erreichen das Gewünschte beim *StatusChangeManager* durch Aufruf seiner Methode `unsubscribe`. Danach ist der Lebenszyklus der Komponente(n) beendet.

6.3.7 Diskussion des SSC-Patterns

An die vorausgegangenen Erläuterungen des SSC-Patterns lässt sich leicht die Frage nach dessen Nutzen anschliessen. Diese Frage soll nun noch eine Beantwortung erfahren.

Der Nutzen des SSC-Patterns liegt darin, dass es eine einfach zu handhabende Möglichkeit bietet, einem der zu Beginn des Kapitels erwähnten Probleme entgegenzutreten: wie kann eine RIA, die zur Laufzeit ja in einer hinsichtlich ihres Funktionsumfangs stets vollständigen und abgeschlossenen Form vorliegt, abhängig von bestimmten Eigenschaften ihre Präsentation an eben diese Eigenschaften anpassen?

Gesucht ist eine gut skalierbare, performante, dabei aber auch einfach zu handhabende Möglichkeit, das Gewünschte zu implementieren. Der notwendige Änderungsaufwand, der dann anfällt,

wenn neue Ereignisse hinzugenommen oder vorhandene entfernt werden sollen, soll möglichst gering sein.

Eine Lösung des dargestellten Problems bei gleichzeitiger Überwindung der vorstehend genannten Hindernisse bietet das SSC-Pattern. Dabei wartet es insbesondere mit den folgenden Vorteilen auf:

- Das "Wissen" über die Menge an Zuständen, die durch eine Komponente überwacht werden, liegt ausschliesslich in der betreffenden Komponente und nicht in der Applikation selber.
- Insbesondere reagiert jede Komponente individuell auf eine eingetretene Zustandsänderung. Auch hier ist es nicht die Applikation, die eine Reaktion auf die Zustandsänderung auslöst, sondern jede Komponente macht dies für sich selber.
- Da jeder zu überwachende Zustand durch ein Interface repräsentiert wird, lässt sich eine beliebige Kombination von zu überwachenden Zuständen erreichen. Auch die Menge an zu überwachenden Zuständen ist insoweit beliebig.
- Das Hinzufügen neuer zu überwachender Zustände ist ohne nennenswerten Aufwand möglich: es muss lediglich das entsprechende Interface als Erweiterung des Interfaces `ISensitive` erstellt werden.
- Einen ebenso geringen Aufwand bedeutet es, Komponenten für bestimmte Ereignisse zu sensibilisieren. Es reicht aus, das entsprechende Interface durch die Komponenten implementieren zu lassen. Umgekehrt bedeutet das: wenn eine Komponente ein Status-Interface *nicht* implementiert, dann liegt auch keine Sensibilisierung vor.
- Wenn eine Änderung an einem überwachten Zustand zur Laufzeit eingetreten ist und eine Komponente benachrichtigt wurde, so erhält sie zugleich auch Kenntnis darüber, *welcher* Zustand sich geändert hat. Dadurch kann sie – sofern sie mehr als einen Zustand überwacht – auch unterschiedlich reagieren.
- Das SSC-Pattern kann wahlweise mit oder ohne Event-Basierung verwendet werden. Der Messaging-Mechanismus wird in der Klasse `StatusChangeManager` gekapselt und erlaubt – sofern die Programmiersprache das *Event Handling* unterstützt – dessen Verwendung, andernfalls aber auch jede andere Benachrichtigungsform. Da der konkret zum Einsatz kommende Messaging-Mechanismus ausschliesslich in der Klasse `StatusChangeManager` notiert ist, kann er dort auch gegen jeden anderen Mechanismus ausgetauscht werden, ohne dass andere Stellen der Applikation modifiziert werden müssen.

Das SSC-Pattern hat sich bereits in der Praxis bewährt, denn es ist Bestandteil der RIA-Version des SLC-Systems an der Fakultät für Mathematik und Wirtschaftswissenschaften der Universität Ulm. Dieses System wird in Kapitel 7 näher vorgestellt, und dort wird auch gezeigt, in welcher Form das SSC-Pattern implementiert wurde (vgl. Abschnitt 7.2.1 auf Seite 189).

6.4 Privilege Aware Menus (Das PAM-Pattern)

Wenn das im letzten Abschnitt vorgestellte SSC-Pattern in einer RIA zur Anwendung kommt, kann potenziell jede in der Anwendung verwendete Komponente auf verschiedene Ereignisse

individuell reagieren. Als ein möglicher Anwendungsfall wurde die Überwachung des Login-Status' vorgeschlagen, sodass sich Komponenten anpassen können, je nachdem, ob sich ein Benutzer an- oder abgemeldet hat.

Zu den Komponenten einer grafischen Benutzeroberfläche, die sich mit Hilfe des SSC-Patterns sensibilisieren lassen, können selbstverständlich auch die Menüstrukturen gezählt werden. Abhängig davon, ob eine Online-Verbindung besteht oder ob ein Benutzer angemeldet ist, können sie ihre Einträge zugänglich machen oder deaktivieren. Jedoch geht eine alleinige Überprüfung daraufhin, *ob* eine Benutzeranmeldung vorliegt oder nicht im Falle der Menüstrukturen nicht weit genug. Denn unterschiedliche Benutzer können durchaus über unterschiedliche Berechtigungen verfügen, sodass die Menüstrukturen nach jeder Änderung des Login-Status' sogleich auch die Berechtigungen des aktuellen Benutzers überprüfen müssen, bevor sie (ggf. weitere) Funktionen zur Verwendung freigeben.

Erneut tritt hier die schon mehrfach genannte Unterscheidung zwischen RIAs und traditionellen Web-Anwendungen zutage: während eine traditionelle Web-Anwendung serverseitig die Menüstrukturen generiert und diese somit stets adäquat zum angemeldeten Benutzer ausliefern kann, verfügt der RIA-Client über die Gesamtfunktionalität der Anwendung, darf aber nur denjenigen Teil seines Funktionskanons zugänglich machen, der den Berechtigungen des aktuell angemeldeten Benutzers entspricht.

Dies führt zu der Frage, wie eine Adaption der Menüstrukturen an Benutzer-Berechtigungen erreicht werden kann. In [HS09a] wurde als Lösungsmöglichkeit für diese Frage das PAM-Pattern vorgeschlagen, das nun im Folgenden ausführlich erläutert wird. PAM steht dabei für *Privilege Aware Menus* und meint Menü- und sonstige Navigationsstrukturen, die über die aktuellen Benutzerberechtigungen ("Privilegien") unterrichtet sind.

6.4.1 Das Konzept hinter dem PAM-Pattern

Das PAM-Pattern setzt voraus, dass der RIA-Client mit einem Backend-Server kommuniziert, welcher die Informationen über die Berechtigungen (im Pattern "Privilegien" bzw. *Privileges* genannt) der einzelnen Benutzer vorhält. Der RIA-Client selbst verfügt über keinerlei derartige Informationen, um keine unnötigen Sicherheitsrisiken zu provozieren (vgl. dazu Abschnitt 6.4.6.2). Sobald Änderungen am Login-Status eintreten, muss der Client zunächst die neue Berechtigungs-Situation überprüfen und darauf geeignet reagieren.

Hier wird deutlich, dass das PAM-Pattern als Ergänzung zum SSC-Pattern des vorigen Abschnitts konzipiert ist: mit Hilfe des SSC-Patterns werden Komponenten über eine Änderung im Login-Status unterrichtet, darunter auch Navigations- und Menükomponenten. Diese sollen mit der Anwendung des PAM-Patterns reagieren, wodurch infolgedessen fortan nur noch die Funktionen zugänglich gemacht werden, die der aktuelle Benutzer aufgrund seiner Berechtigungen verwenden darf.

Damit diese Reaktionen implementiert werden können, ist ein Mechanismus erforderlich, um die Privilegien zu repräsentieren und sie zwischen Client und Server zu kommunizieren. Hier bieten sich abermals XML- und JSON-Notationen an, aber auch Remoting-Mechanismen (vgl. dazu auch die Ausführungen in Kapitel 5). In Bezug auf Sicherheit stellt der Austausch der Privilegien zwischen Client und Server einen hochsensiblen Bereich dar, denn er ist ein potenzieller Angriffspunkt für Kompromittierungsversuche (siehe dazu auch Abschnitt 6.4.6.2). Das PAM-Pattern abstrahiert an dieser Stelle vom tatsächlich verwendeten Daten- bzw. Objektaustausch-Mechanismus und macht somit Raum für dessen individuelle Ausgestaltung.

Damit die Menüstrukturen[2] auf den Privilegien-Status reagieren können, muss die RIA Kenntnis darüber besitzen, welche Funktionen welche Privilegien voraussetzen. Auch auf diese Frage hält das PAM-Pattern eine Antwort bereit.

Der Gedanke hinter dem PAM-Pattern ist, dass sich die RIAs aus einer Menge von funktionalen Einheiten ("*Functional Entities*") zusammensetzen, die wie folgt definiert sind:

Definition: Funktionale Einheit (engl.: *Functional Entity*) Eine "funktionale Einheit" ist eine einzelne Funktionalität oder eine Menge einzelner Funktionalitäten einer RIA, die unabhängig voneinander existieren. Eine funktionale Einheit kann durch einen oder verschiedene Einträge in die Navigationsstrukturen der Anwendung aufgerufen werden und stellt sich daraufhin selbstständig in dem dafür vorgesehenen Bereich der Anwendung dar.

Diese Definition lässt erkennen, dass funktionale Einheiten mit den in Abschnitt 7.1.3.2 auf Seite 185 definierten Portlets vergleichbar sind.

Das Ziel hinter dem PAM-Ansatz ist es demnach, dass

- jede funktionale Einheit weiß, welche Privilegien sie voraussetzt,
- sie dieses Wissen an die Menüstruktur(en) kommuniziert
- und jene Strukturen dann nur diejenigen funktionalen Einheiten zugänglich machen, die den jeweiligen Benutzerprivilegien adäquat sind.

Aus diesen Zielen resultieren die folgenden Schritte zur Anwendung des PAM-Patterns:

1. Codieren der Privilegien,
2. Anfragen aller funktionalen Einheiten, welche Privilegien sie voraussetzen und
3. Anpassen der Menüstrukturen anhand der Benutzerberechtigungen (nach jedem erfolgten Login).

Das UML-Sequenzdiagramm in Abbildung 6.4 fasst das Konzept des PAM-Ansatzes zusammen und visualisiert den Ablauf.

Erläuterungen zu Abbildung 6.4: Beim Starten der Applikation ("*RIA Core*") kann noch keine Anmeldung erfolgt sein. Deshalb wird die RIA zunächst mit dem Privilegienwert 0 initialisiert, der angibt, dass der momentane Benutzer keinerlei Privilegien besitzt. Daraufhin erstellt die Anwendung die Menüstruktur(en), deren (auf funktionale Einheiten verweisende) Einträge sich selbstständig darüber informieren, welche Berechtigungen notwendig sind, um die jeweiligen funktionalen Einheiten auszuwählen. Wenn sich ein Benutzer erfolgreich angemeldet hat, erhält die RIA vom Backend-Server Kenntnis über die dem Benutzer zugeteilten Privilegien, und die Menüstruktur(en) passt/passen sich an die neue Berechtigungssituation an. Ähnlich ist der Ablauf dann, wenn sich der Benutzer wieder abmeldet: in diesem Fall verfügt er – wie zu Beginn der Anwendung – über keinerlei Berechtigungen, was dem Privilegienwert 0 entspricht. Erneut passen sich die Menüstrukturen selbstständig an diese neue Situation an.

[2]Wenn im folgenden der Begriff "Menüstrukturen" verwendet wird, so meint er Navigationsstrukturen der RIA allgemein, nicht nur jene, die durch ein Menü repräsentiert werden.

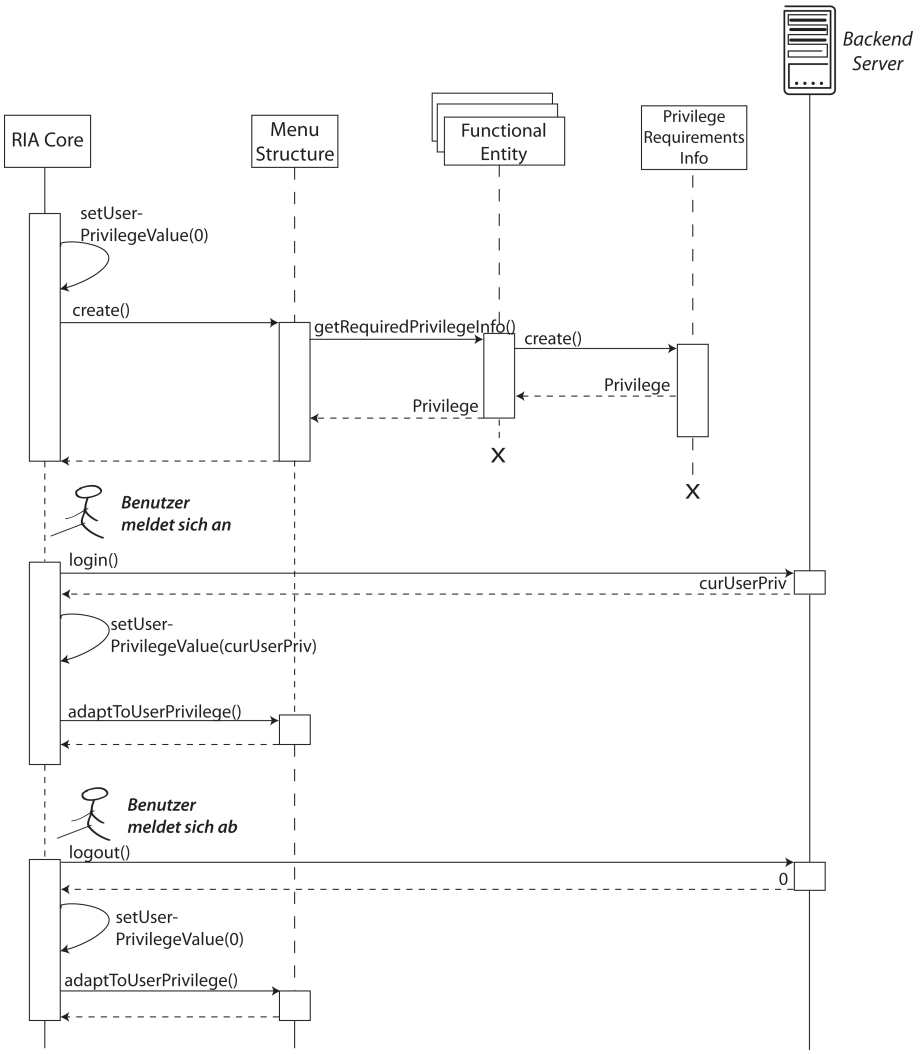


Abbildung 6.4: Laufzeitverhalten des PAM-Patterns

6.4.2 Codieren der Privilegien

Das PAM-Pattern sieht die Verwendung einer Klasse namens `Privilege` vor. Instanzen dieser Klasse repräsentieren Berechtigungen und stellen Methoden zur Verfügung, um diese mit anderen Privilegien (also mit anderen Objekten dieser Klasse) zu vergleichen. Grundsätzlich kommen zwei mögliche System infrage, um die Berechtigungen abzubilden:

hierarchische Systeme: Die Berechtigungen bilden eine Hierarchie, d.h. eine Berechtigung auf einer bestimmten Stufe schließt automatisch alle Berechtigungen früherer Stufen mit ein.

kombinatorische Systeme: Jede Berechtigung wird einzeln betrachtet, und einem Benutzer wird eine individuelle Berechtigungs-Kombination exakt zugeteilt.

Die letztendlich verwendete Systematik bleibt aus Sicht des PAM-Patterns allerdings verborgen, denn das Pattern kapselt den zugrunde liegenden Mechanismus der Privilegien-Codierung in der oben genannten Klasse `Privilege`. Somit kann ein Mechanismus jederzeit ausgetauscht

werden, ohne Änderungen an anderer Stelle als in der Klasse `Privilege` erforderlich zu machen.

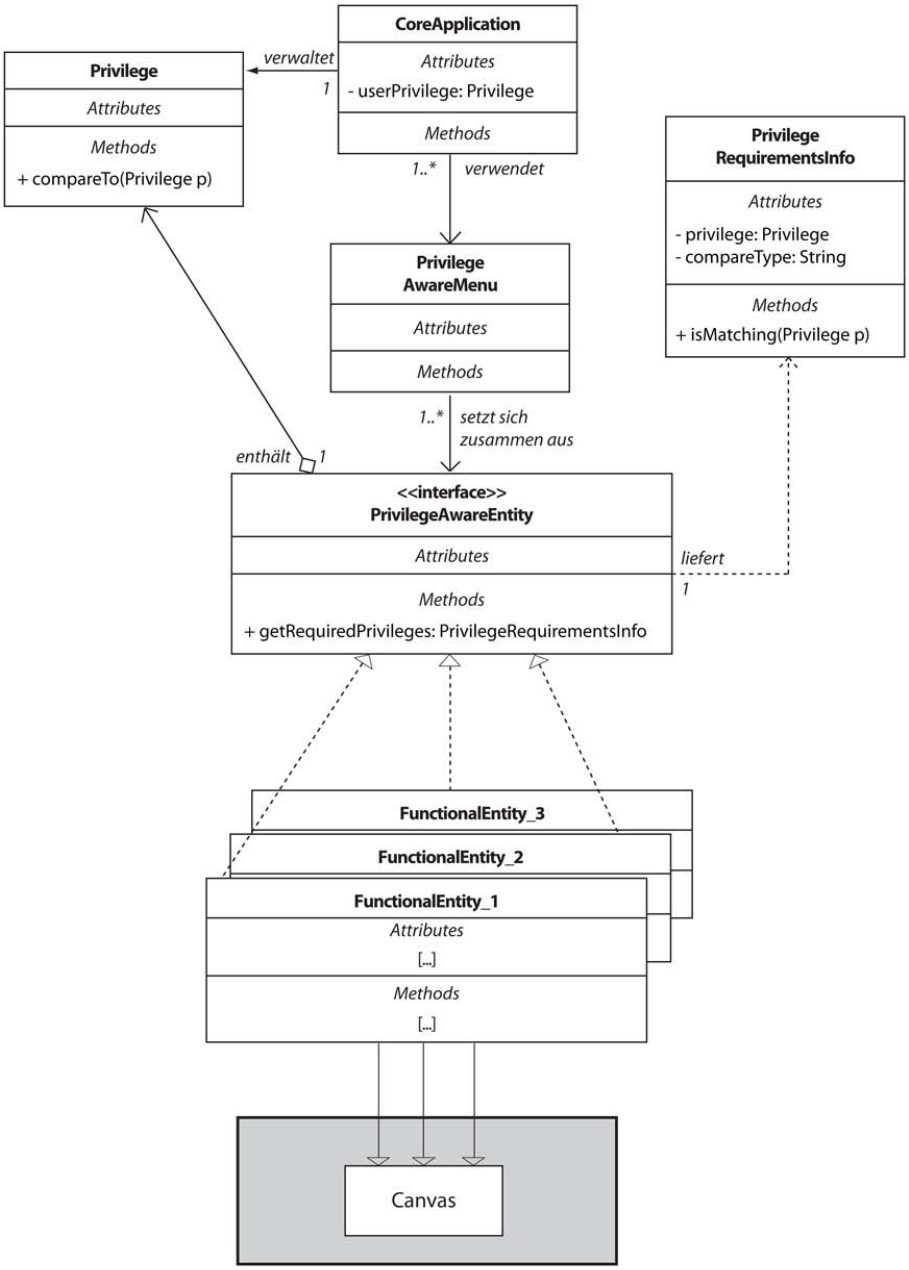


Abbildung 6.5: UML-Klassendiagramm des PAM-Patterns

In Abbildung 6.5 ist das PAM-Pattern in Form eines UML-Klassendiagramms dargestellt. Aus ihm wird zunächst ersichtlich, dass der Kern der RIA (jener Teil, der die Navigations- bzw. Menüstrukturen verwaltet) Kenntnis über den Berechtigungsumfang des aktuell angemeldeten Benutzers hat. Dazu zählt insbesondere auch der Zustand, dass ein Benutzer *nicht* angemeldet ist und demzufolge keinerlei Berechtigungen vorhanden sind. Die durch den Kern der RIA verwalteten Menüstrukturen setzen sich aus Einträgen zusammen, die das Interface

`PrivilegeAwareEntity` implementieren. Dieses sowie die übrigen im Pattern verwendeten Klassen werden in den nachfolgenden Abschnitten erläutert.

6.4.3 Das Interface `PrivilegeAwareEntity`

Das PAM-Pattern sieht vor, dass nicht die Anwendung als solche steuert, welche Funktionen welche Berechtigungen voraussetzen, sondern dass jede funktionale Einheit einer RIA diese Information über und bei sich selber führt. Dies macht eine Anwendung sehr gut skalierbar. Um eine funktionale Einheit am Berechtigungssystem des PAM-Patterns teilnehmen zu lassen, muss sie ein Interface namens `PrivilegeAwareEntity` implementieren. Dieses beinhaltet lediglich eine öffentliche Methode (`getRequiredPrivileges()`), mit Hilfe derer die von dieser Einheit benötigten Berechtigungen "gemeldet" werden können. Als Informationsträger wird hierbei ein Objekt der im folgenden Unterabschnitt beschriebenen Klasse `PrivilegeRequirementsInfo` gebraucht.

6.4.4 Die Klasse `PrivilegeRequirementsInfo`

Der Rückgabewert der zuvor genannten Methode `getRequiredPrivileges()` besteht aus einem Objekt der Klasse `PrivilegeRequirementsInfo`. Diese Objekte beinhalten zwei Informationen:

- die benötigten Berechtigungen (ausgedrückt mit Hilfe von Objekten der Klasse `Privilege`) sowie
- einen Vergleichsmaßstab ("genau", "mindestens", "höchstens" etc.)

Darüberhinaus stellen diese Objekte eine öffentliche Methode `isMatching` zur Verfügung, die als Parameter die Berechtigungen des aktuellen Benutzers übergeben bekommt (ebenfalls in Form eines Objektes der Klasse `Privilege`), und prüft, ob diese Berechtigungen den Anforderungen genügen. Der Rückgabewert dieser Methode ist vom Typ `boolean` und gibt `true` zurück, wenn die Berechtigungen den Anforderungen genügen, andernfalls `false`.

6.4.5 Die Klasse `PrivilegeAwareMenu`

Mit den bislang vorgestellten Klassen bzw. Interfaces und deren Interaktion untereinander steht eine Möglichkeit bereit, mit Hilfe derer die funktionalen Entitäten ihre benötigten Berechtigungen zum Ausdruck bringen können. Es soll nun die ergänzende Erklärung folgen, wie die derartigen Informationen verwendet werden. Dabei erlaubt das PAM-Pattern seine Anwendung auf potenziell allen grafischen Komponenten. Das angedachte Szenario sieht indes eine Verwendung in Menüstrukturen vor. Dies soll an einem Beispiel nachstehend erläutert werden.

Wenn die RIA gestartet wird, muss zunächst eine Menüstruktur aus den zur Verfügung stehenden funktionalen Einheiten aufgebaut werden. In RIAs ist zu diesem Zweck die Verwendung von XML-Dokumenten durchaus üblich. Wie sie gewonnen werden, soll dabei zunächst von den Erläuterungen ausgenommen bleiben. Denkbar sind hierzu beispielsweise Adaptionen des in [Hei04a] beschriebenen Vorgehens. Stattdessen fokussiert dieser Abschnitt auf die Erweiterungen, die an XML-Dokumenten vorgenommen werden müssen, um sie im Sinne des PAM-Patterns "aufmerksam" für Berechtigungen – mithin also im Sinne der Namensgebung

des Patterns *privilege aware* – zu machen. Listing 6.4 zeigt einen Ausschnitt aus einem entsprechenden XML-Dokument.

Listing 6.4: Ausschnitt aus einer Menübeschreibung in XML

```
1 <menu>
2     <menuitem label="Get Grades" className="getGradePanel" actionTag=""/>
3     <menuitem label="Enter Grades" className="enterGradePanel" actionTag=""/>
4     ...
5 </menu>
```

6.4.5.1 Erläuterungen zu Listing 6.4

Listing 6.4 zeigt drei Attribute, die jeder Menüeintrag haben muss:

`label`: gibt die Bezeichnung an, unter der sich der Eintrag in die Menüstruktur einbringt,

`className`: referenziert die Klasse, welche die entsprechende Funktionalität bietet (funktionale Einheit),

`actionTag`: dieses Attribut wird aus der RIA heraus gesetzt und ausgewertet; es dient dazu, Menüeinträge, die auf ein bestimmtes Ereignis reagieren sollen, zu kennzeichen.

Listing 6.5 gibt ein Codebeispiel für den angedachten Umgang mit XML-Menüstrukturen.

Listing 6.5: Beispiel für das Entfernen von Einträgen in der Menüstruktur aus Listing 6.4

```
1  public function adaptMenuStructure {
2   for each(item in menu..menuitem) {
3    if (!getRequiredPrivilegeInfo(item.@className).isMatching(userPrivilege))
4     {item.@actionTag="detach"; //mark for removal}
5   }
6   var candidates:XMLList = menu..menuitem.(@actionTag=="detach");
7   for (var i:int=candidates.length()-1; i >= 0; i--) {
8    delete candidates[i];
9   }
10 }
```

Erläuterungen zu Listing 6.5: In den Zeilen 2 bis 5 wird für jeden Eintrag der Menüstruktur die Methode `getRequiredPrivilegeInfo()` aufgerufen. Sie erfragt bei der dem jeweiligen Menüeintrag zugeordneten funkionalen Einheit (repräsentiert durch die im Attribut `className` angegebene Klasse) den zum Gebrauch der funktionalen Einheit notwendigen Berechtigungsstatus. Jede der in den Menüeinträgen angegebenen Klassen muss eine Implementierung des Interfaces `PrivilegeAwareEntity` sein, damit der Aufruf der Methode `getRequiredPrivileges()` gelingen kann. Diese Methode gibt eine Instanz der in Abschnitt 6.4.4 beschriebenen Klasse `PrivilegeRequirementsInfo` zurück, welche sich dann mit dem Berechtigungsstatus des aktuell angemeldeten Benutzers vergleichen lässt (vgl. Objektvariable `userPrivilege`). Reichen die Nutzerprivilegien nicht aus, so wird der Menüeintrag als zu entfernen gekennzeichnet (Zeile 4). Zeile 6 führt zu einer Datenstruktur, die *nur* aus denjenigen Einträgen der Menüstruktur besteht, die zuvor als zu entfernen gekennzeichnet wurden (an dieser Stelle wird die Bedeutung des Attributs `actionTag` erkennbar). Die somit gewonnene Datenstruktur wird in den Zeilen 7 bis 9 komplett durchlaufen, um auf diese Weise jeden Eintrag einzeln zu

löschen. Weil an dieser Stelle mit Referenzen anstelle von Kopien gearbeitet wird, bedeutet dies, dass zugleich auch die Einträge in der Original-Struktur gelöscht werden.

Mithin enthalten die Zeilen 6 bis 9 einen einfachen, aber sehr effektiven Weg, um einzelne Einträge aus der gesamten Menüstruktur zu löschen. Dabei muss jedoch darauf geachtet werden, dass sich die Original-Struktur zu einem späteren Zeitpunkt wieder restaurieren lässt. Dies ist im einfachsten Falle dadurch möglich, dass sie nach Erstellung der Struktur beim Start der Anwendung in einer separaten Variable stets als Kopie für diese Zwecke gespeichert wird.

6.4.6 Diskussion des PAM-Patterns

Das PAM-Pattern ist dazu gedacht, immer dann zum Einsatz zu kommen, wenn eine RIA als Frontend-System verwendet wird, das – abhängig von den Berechtigungen des aktuellen Benutzers – nur eine Teilmenge der insgesamt zur Verfügung stehenden Funktionalitäten präsentieren soll. Die denkbaren Beispiele für den Einsatz reichen dabei von Web-Anwendungen bis hin zu Fernwartungs-Aktivitäten auf entfernten Systemen. Ein reales Anwendungsszenario des PAM-Pattern folgt im nachstehenden Kapitel.

6.4.6.1 Bezug zum SSC-Pattern

Seine Stärke spielt das PAM-Pattern in Verbindung mit dem in Abschnitt 6.3 (Seite 161) vorgestellten SSC-Pattern aus. In diesem Fall sähe das typische Szenario so aus, dass im ersten Schritt die Menükomponente hinsichtlich des Login- und des Online-Status' sensitiv gemacht wird. Ändert sich zur Laufzeit der Login- und/oder der Online-Status, so kann die Menükomponente darauf geeignet reagieren, indem sie – als *Privilege Aware Menu Structure* – die enthaltenen Einträge der neu eingetretenen Berechtigungssituation anpasst.

Zeitlich betrachtet, greift das SSC-Pattern also (unmittelbar) vor dem PAM-Pattern, denn letzteres dient ja als Reaktion auf Änderungen im Berechtigungsstatus, die typischerweise nur durch eine vorhergehende Änderung im Login- oder Online-Status eintreten können.

Es ist deshalb empfehlenswert, beide Pattern zugleich in einer Anwendung zu implementieren, da sie gemeinsam ihre jeweilige Stärke besser ausspielen können.

6.4.6.2 Eine Bemerkung zur Sicherheit

Die Anwendung des PAM-Patterns auf der Client-Seite darf nicht als Ersatz für eine serverseitige Sicherheitsüberprüfung verstanden werden. Das PAM-Pattern dient als Ergänzung einer solchen und trägt dazu bei, den Gedanken der *Individualisierung* von Rich Internet Applications zu realisieren, indem es Menü- und Navigationsstrukturen adäquat zu den Berechtigungen des aktuellen Benutzers gestaltet. Die Verantwortung, dass die entsprechenden serverseitigen Funktionen nur dann ausgeführt werden können, wenn sich ein über ausreichende Berechtigungen verfügender Benutzer zuvor erfolgreich authentifiziert hat, trägt letztendlich der Server. Denn ein RIA-Client kann manipuliert werden, ebenso ist dies für die Kommunikation zwischen Client und Server denkbar. Eine serverseitige Sicherheitsüberprüfung ist daher unabdingbar.

6.5 Multilanguage Components

Die Unterstützung von Mehrsprachigkeit in serviceorientierten RIAs unterscheidet sich nicht von derjenigen in inhaltszentrierten RIAs. Deshalb stehen dieselben Lösungen zur Verfügung, die in Zusammenhang mit inhaltszentrierten RIAs vorgestellt wurden. Insbesondere kann selbstverständlich auch das Konzept des MultiLanguageLabelers (MLL), das im Abschnitt 5.3.3 auf Seite 122 präsentiert wurde, verwendet werden.

Wird eine RIA zum Einsatz in einem Stand-Alone-Player anstelle eines Browser-Plug-ins entwickelt, lässt sich das Konzept des MultiLanguageLabelers sogar noch erweitern. Denn in diesem Fall stehen auch lokale Ressourcen (insbesondere das Dateisystem) bereit, um Sprachdefinitionsdateien[3] schnell verfügbar zu haben und nicht erst von einem entfernten Server beziehen zu müssen. Somit ist auch eine mehrsprachige Verwendung der RIA im Offline-Modus problemlos möglich.

Bei der Verwendung von Adobe AIR als Zielplattform steht auf dem Zielrechner eine lokale Datenbank (SQLite) zur Verfügung, welche sich – neben dem Dateisystem – ebenfalls zur Persistierung von Sprachdefinitionen eignet. In diesem Fall erfolgt die Auslieferung der sprachspezifischen Inhalte nicht durch den Server, sondern durch die lokale Datenbank.

Mithin ergeben sich mehrere verschiedene mögliche Quellen für den Bezug der Sprachdefinitionen, sodass das MLL-Pattern entsprechend modifiziert werden muss. Aufgrund dessen Konzeption lässt sich die notwendige Flexibilisierung des Bezugsmechanismus' für Sprachdefinitionen sehr leicht erhalten, denn es muss lediglich die in Abschnitt 5.3.1 auf Seite 118 beschriebene Abstraktion des Inhaltsbezugs im RAP-Pattern adaptiert werden. Genau wie in Zusammenhang mit dem Bezug der Inhalte aus verschiedenen Quellen beschrieben, wird dann auch das MLL-Pattern gegen ein Interface entwickelt und nicht gegen einen konkreten Bezugsmechanismus, sodass dabei ein ganz ähnliches Klassendiagramm wie jenes in Abbildung 5.3 auf Seite 119 entsteht. An die Stelle der Remoting- bzw. Servlet-Klassen aus jener Abbildung treten nun die verschiedenen Bezugsarten der Sprachdefinitionen (via Dateisystem, via Datenbank-Anbindnung oder abermals via Servlet/HTTP).

6.5.1 Querbezug von MLL- und SSC-Pattern

Neben der Wiederverwendbarkeit des Zugriffsabstraktionsprinzips profitiert der MLL-Ansatz in Stand-Alone-RIAs aber auch von dem zu Beginn dieses Kapitels beschriebenen SSC-Pattern. Dann nämlich, wenn auch ein Offline-Betrieb der RIA vorgesehen ist, ergänzen sich die Konzepte wie folgt:

1. Bei der erstmaligen Verwendung der RIA wird eine Liste aller unterstützten Sprachen vom Server bezogen und lokal persitiert. (Ob in der SQLite-Datenbank oder im Dateisystem ist dabei unerheblich.) Man beachte, dass hier nicht die Sprachdefinitionsinformationen selber gemeint sind, sondern lediglich eine Auflistung der unterstützten Sprachen.

2. Die tatsächlichen Sprachdefinitionen der Standard-Sprache oder – sofern der Benutzer diese umgestellt hat – der aktuell gewählten Sprache werden ebenfalls bezogen und lokal persistiert.

[3] Dieser Begriff wurde in Abschnitt 5.3.3 erstmals verwendet, und Listing 5.7 auf Seite 124 zeigte eine solche Datei. Im aktuellen Kontext handelt es sich jedoch nicht mehr notwendigerweise um eine Sprachdefinitions*datei*, sodass nun der allgemeinere Begriff Sprachdefinitions*infomation* oder lediglich der Terminus *Sprachdefinition* benutzt wird.

3. Fortan steht die RIA offline in der gewünschten Sprache zur Verfügung.
4. Durch entsprechende Anwendung des SSC-Patterns kann erreicht werden, dass die RIA – sobald sie eine bestehende Online-Verbindung detektiert – beim Server prüft, ob dort zwischenzeitlich eine aktuellere Sprachdefinition als die lokal verwendete für die ausgewählte Sprache bereitsteht.
5. Ebenso kann bei bestehender Online-Verbindung die Sprache umgestellt werden, was (analog zu Schritt 2) den Bezug einer weiteren Sprachdefinition nebst deren lokaler Persistierung zur Folge hat.
6. Fortan stehen also auch lokal mehrere Sprachversionen zur Verfügung und erlauben ein Umschalten zwischen diesen Sprachen auch im Offline-Betrieb.
7. Mit jedem Herstellen einer Online-Verbindung wiederholt sich das Vorgehen gemäß den Punkten 4 und 5.

Damit je nach Betriebsart (online oder offline) die Bezugsart der Sprachdefinitionsinformationen flexibel variiert werden kann, ist die eingangs erwähnte Zugriffsabstraktion sehr hilfreich, denn sie erlaubt (durch die Verwendung eines Interfaces), den Programmcode unabhängig von der konkreten Bezugsart zu formulieren. Erst zur Laufzeit werden dann – je nach Erfordernis – wahlweise die Online- oder die Offline-Implementierung des Interfaces *konkret* verwendet. Erneut findet an dieser Stelle das bereits in Abschnitt 5.3.1 auf Seite 118 erwähnte Prinzip der *Inversion of Control* Anwendung.

Weil die hier beschriebene Idee des Sprachinformationsbezugsmechanismus' eine Adaption der in Abschnitt 5.3.1 beschriebenen Idee darstellt, soll auf eine Darstellung eines Klassendiagramms an dieser Stelle verzichtet werden, denn dieses entspräche im Wesentlichen dem Diagramm aus Abbildung 5.3 auf Seite 119.

6.5.2 Update-Detektion

Das beschriebene Vorgehen bildet einen Auto-Update-Mechanismus nach: im Offline-Betrieb steht die zuletzt vom Server bezogene (Sprach-)Version zur Verfügung. Sobald aber eine Online-Verbindung unterhalten wird, besteht die Möglichkeit, Updates (hier: Sprach-Updates) zu beziehen.

Sehr leicht lässt sich aus diesem Ablauf auch ein tatsächlicher Update-Mechanismus (also einer, der nicht für die Erkennung neuer Sprachversionen, sondern für die Erkennung einer aktuelleren als der momentan verwendeten RIA-Version vorgesehen ist) erhalten. Durch die Anwendung des SSC-Patterns kann dafür Sorge getragen werden, dass die RIA (je nach Architektur der RIA auch: einzelne Komponenten) geeignet reagiert (reagieren), wenn die Anwendung (oder Teile davon) nicht in der erforderlichen Version vorliegt. Ein Beispiel für eine konkrete Verwendung eines solchen Update-Mechanismus' folgt in dem mit "Versionskontrolle" überschriebenen Abschnitt 7.2.4 auf Seite 197.

6.6 Zusammenfassung zur Laufzeit-Personalisierung dynamischer RIAs

In diesem Kapitel konnte gezeigt werden, wie mit Hilfe der vorgestellten Entwurfsmuster (*Design Pattern*) Laufzeit-Anpassungen der RIA an Benutzer-Präferenzen, an Benutzer-Privilegien

und an Änderungen "äußerer Umstände", wie beispielsweise Änderungen im Konnektivitäts-Status und/oder im Login-Status sehr einfach gehandhabt werden können. Während die einzelnen Pattern durchaus isoliert voneinander verwendet werden können und sich gegenseitig nicht bedingen, spielen sie ihr Potenzial insbesondere dann aus, wenn sie gemeinsam in einer RIA verwendet werden und sich insoweit untereinander ergänzen.

Allen hier präsentierten Pattern ist gemein, dass sie dazu führen, dass nicht die RIA selbst, sondern ihre (funktionalen) Bestandteile über bestimmtes "Wissen" (beispielsweise über die zu ihrer Benutzung erforderlichen Berechtigungen oder über die auszuführenden Aktionen, wenn bestimmte Situationen oder Ereignisse eintreten) verfügen. Somit führen die Pattern zu einer flexibel erweiterbaren und vor allem gut skalierbaren RIA.

Während die in diesem Kapitel vorgenommenen Erläuterungen ausschließlich abstrakter Natur waren und ohne Veranschaulichung anhand eines Beispiels erfolgten, präsentiert das nächste Kapitel ein reales Softwaresystem, in dessen RIA-Client alle hier vorgestellten Pattern erfolgreich implementiert werden konnten. Neben der Beschreibung und Einordnung des Gesamtsystems dient das nachfolgende Kapitel damit vor allem dazu, die Praxistauglichkeit der Pattern unter Beweis zu stellen.

Kapitel 7

Anwendungsbeispiel: SLC on AIR

"Die Maxime des ökonomischen Entwerfens heisst Entworfenes wiederverwenden"[QC99]

7.1 Überblick über das SLC-System

Das vorstehende Kapitel zeigte verschiedene Problemstellungen in Zusammenhang mit RIAs auf und schlug Entwurfsmuster zu deren Lösung vor, verzichtete jedoch auf konkrete praxisnahe Beispiele. Diese sollen im vorliegenden Kapitel nun folgen und somit unter Beweis stellen, dass die diskutierten Ansätze praxistauglich sind und dies auch als Bestandteile eines im Produktivbetrieb eingesetzten Softwaresystems demonstrieren.

Als entsprechendes Softwaresystem dient für alle vorgestellten Pattern das "SLC-β-System" der Fakultät für Mathematik und Wirtschaftswissenschaften an der Universität Ulm, das gleichzeitig mit der vorliegenden Arbeit entstanden ist und seinen rund 8 Jahre alten Vorgänger (Projektname "SLC") ablöste.

In diesem Kapitel wird zunächst ein kurzer Überblick über die bisherige Version des SLC-Systems gegeben, dem eine Vorstellung der Architektur und der verwendeten Technologien im neuen System folgt. Bestandteil dieses Systems ist ein RIA-Client, auf den die in dieser Arbeit präsentierten Pattern erfolgreich angewendet wurden. Mithin wird im weiteren Verlauf dieses Kapitels präsentiert, in welcher Form die Pattern auf den RIA-Client von SLC-β übertragen wurden. Mit einem Erfahrungsbericht und dem Ausblick auf weitere Entwicklungsschritte im SLC-β-System schliesst dieses Kapitel ab.

7.1.1 SLC: Vorstellung und Entwicklungshistorie

Im Jahr 2000 wurde an der Fakultät für Mathematik und Wirtschaftswissenschaften der Universität Ulm ein Softwaresystem in Betrieb genommen, das dort zur Verwaltung studentischer Belange (z.B. Anmeldung zu Vorlesungen, Verwaltung erreichter Übungs- und Klausurergebnisse) sowie darüberhinaus auch für administrative Aufgaben im Lehrbetrieb (z.B. Vorlesungsplanung) oder im Rechnerbetrieb (Verwaltung von Benutzeraccounts für das Rechner-Netzwerk) dient. Dieser Einsatzzweck des Software-Systems war zugleich auch dessen Namensgeber: das System wurde auf den Namen SLC getauft, wobei diese drei Buchstaben ein Akronym der Begriffe "Students, Lectures, Courses" darstellen.

SLC ist aus der wissenschaftlichen Beschäftigung mit der damals aufkommenden Portal-Technologie entstanden (vgl. [Mel02]) und wurde deshalb als Portal konzipiert und implementiert.

Für rund 8 Jahre war SLC nahezu störungsfrei in der Fakultät im Einsatz und wurde in dieser Zeit sukzessive um viele verschiedene Funktionen erweitert. Aufkommende Wünsche nach Erweiterungen zeigten in letzter Zeit aber auch die Grenzen der verwendeten Technik auf. Insbesondere der Wunsch, mittelfristig auch multimediale e-Learning-Funktionalitäten mit Hilfe des SLCs zu ermöglichen, führten letztendlich dazu, eine Nachfolger-Version zu konzipieren, welche technisch auf dem neuesten Stand und für die nächsten Jahre zukunftsfähig sein sollte.

Unter der Bezeichnung SLC-β wurde diese neue Version des SLC innerhalb des Instituts für Angewandte Informationsverarbeitung (IAI) der Universität Ulm in der Zeit von Oktober 2007 bis Oktober 2008 entwickelt und konnte dann planmäßig zu Beginn des Wintersemesters 2008/2009 in Betrieb genommen werden. Anders, als es die gewählte Bezeichnung suggeriert, handelt es sich bei SLC-β nicht um eine Software, die in einer noch nicht fertigen Beta-Version vorliegt, vielmehr soll das "Beta" im Sinne von "zwei" (als zweiter Buchstabe des griechischen Alphabets) verstanden werden und somit eine neue Generation des Software-Systems proklamieren.

Die Entwicklung von SLC-β verfolgte von Anfang an zwei Ziele: einerseits sollte es wieder eine klassische Web- bzw. Portallösung geben, welche als unmittelbarer Ersatz für die bisherige Version konzipiert war. Andererseits wurde zu SLC-β aber auch gleichzeitig ein RIA-Client entwickelt, der seine Stärken insbesondere in der multimedialen – *reichhaltigen* – Präsentation (mithin also in den angedachten e-Learning-Komponenten des neuen Systems) haben sollte. Insofern wurde also eine strikte Unterteilung in Geschäftslogik (Backend-System) und Präsentationslogik (Frontend) vorgenommen. Beide Teile werden weiter unten noch näher vorgestellt. Konkret bedeutete diese Unterteilung, dass *ein* Backend und *zwei* Frontends entwickelt werden mussten. Mit jedem der beiden Frontends wurden jedoch unterschiedliche Ziele verfolgt:

Das web-basierte Frontend

Das webbasierte Frontend war stets als unmittelbarer Ersatz für die bisherige SLC-Version angedacht. Es stellt den vollständigen Funktionsumfang zur Verfügung und dient somit nicht nur Studierenden, sondern auch Mitarbeitern z.B. für die Vorlesungsverwaltung, Benutzerverwaltung und verwandte administrative Aufgaben. Die Präsentation erfolgt durch reines HTML (d.h. ohne dynamische Anreicherungen beispielsweise durch AJAX), sodass dieses Frontend im Sinne der Diskussion in Abschnitt 5.1.2.1 als barrierefreie Fallback-Lösung für die Funktionen des RIA-Clients verstanden werden kann.

Das RIA-Frontend

Als Zielgruppe des RIA-Frontends wurden die studentischen Benutzer identifiziert, nicht hingegen Lehrende oder das Administrations-Personal. Deshalb ist der Funktionsumfang des RIA-Frontends auf diejenigen Funktionen eingegrenzt, die für Studierende interessant sind. Es ist darauf ausgelegt, neben den traditionellen SLC-Funktionen für Studierende auch die mittelfristig angedachten multimedialen e-Learning-Funktionalitäten bereitzustellen. Insbesondere soll die RIA-Version aber auch auf mobilen Geräten (z.B. Mobiltelefonen) lauffähig sein und somit eine "ubiquitäre" Verwendung des SLC-β-Systems ermöglichen.

Für diese Arbeit von besonderem Interesse sind einerseits der Teilbereich des Backends, der für die Kommunikation mit dem RIA-Client zuständig ist, sowie andererseits der RIA-Client selbst. In letzterem wurden die in Kapitel 6 vorgestellten Pattern erfolgreich umgesetzt. Die nächsten Abschnitte stellen Back- und Frontends detaillierter vor und gehen dabei auf die genannten bedeutsamen Punkte besonders ein.

7.1.2 Backendtechnologien in SLC-β

Die für die Entwicklung des Backends von SLC-β eingesetzten Frameworks und Technologiekonzepte wurden nicht danach ausgewählt, eine besonders enge Zusammenarbeit mit Flex-Clients zu ermöglichen. Im Gegenteil: einer der Vorteile der in dieser Arbeit vorgestellten Konzepte ist gerade ihre Allgemeingültigkeit, d.h. ihre Unabhängigkeit von konkreten Backend-Technologien. Die nachstehende Auflistung der wichtigsten Backend-Technologien von SLC-β sei somit nur aus Gründen der Vollständigkeit erwähnt.

CXF: Die Kommunikation zwischen Backend und RIA-Client erfolgt in SLC-β über Web-Services. Als Service-Framework kommt dazu das CXF-Projekt der Apache Foundation[1] zum Einsatz. Dies insbesondere deshalb, weil es eine native Unterstützung des Spring Frameworks (siehe nächster Punkt) anbietet.

Spring Framework: Eine eingehende Erklärung des Spring Frameworks[2] würde den Rahmen dieser Arbeit sprengen und zudem nichts Substantielles beitragen. Aus diesem Grunde sei lediglich vermerkt, *dass* SLC-β auf dem Spring Framework basiert und von dessen Programmierparadigmen sowie der hohen Integration der beteiligten Technologien (Persistenzschicht, Web-Integration etc.) profitiert. Insbesondere ermöglichte die Verwendung von Spring (und einiger dessen Erweiterungen) einen hochgradig modularen Aufbau von SLC-β, der zudem ein sehr enges Zusammenspiel mit OSGi (siehe nächster Punkt) erlaubt.

OSGi: Das Backend ist modular aufgebaut, und einzelne Komponenten sollen dynamisch zur Laufzeit hinzugenommen oder auch entfernt werden können, ohne damit andere Teile des Systems zu beeinflussen. Das bedeutet, dass SLC-β die typischen Eigenschaften eines Plug-in-Systems besitzen soll. Alleine, um dieses Ziel zu erreichen, ist der Einsatz von OSGi noch nicht zwingend nötig, jedoch erlaubt OSGi eine feingranulare Steuerung, welche Komponenten auf welche Ressourcen zugreifen dürfen und "zwingt" den Entwickler zu einem Programmierstil, der diese Steuerung nicht unterminieren kann. OSGi kommt in SLC-β deshalb vor allem im Interesse einer langfristigen Wart-/Pflegbarkeit des Systems zum Einsatz.

eclipseLink: Als Persistenzframework wird *eclipseLink*[3] verwendet. Da dies für die Gestaltung des RIA-Clients jedoch ohne Bedeutung ist, kann auf eine nähere Vorstellung dieses Frameworks verzichtet werden.

7.1.3 Das Web-Frontend in SLC-β

Da es sich beim gesamten SLC-β-System um eine Web-Applikation handelt, sind die anzuzeigenden Inhalte (unabhängig davon, ob sie als HTML-Version im Browser oder in Form einer

[1] siehe `http://cxf.apache.org/`
[2] siehe `http://www.springsource.org/`
[3] siehe `http://www.eclipse.org/eclipselink/`

reichhaltigen RIA-Variante ausgegeben werden sollen) ausschließlich interaktionszentrierter Natur (vgl. dazu die Definition auf Seite 79). Deshalb greifen die in Kapitel 5 beschriebenen Ansätze, um eine automatische barrierefreie HTML-Fallback-Variante der Applikation zu generieren, nicht. Überdies hat die Diskussion in Abschnitt 4.5 ergeben, dass RIAs zum Zeitpunkt der Erstellung dieser Arbeit noch nicht vollständig barrierefrei gestaltet werden können. Weil SLC-β an einer öffentlichen Einrichtung betrieben wird, ist es nach Paragraph 11, Abschnitt 1, BBG jedoch von Bedeutung, dass ein barrierefreier Zugang zu diesem System möglich ist. Deshalb – und natürlich aber auch, weil (wie beschrieben) der RIA-Client nur auf die Verwendung durch die Studierenden ausgerichtet ist und somit nicht über Funktionen für Lehrende und das Administrationspersonal verfügt – existieren zwei Frontends zu SLC-β.

Die im Mittelpunkt dieser Arbeit stehenden Pattern sind zwar zur Anwendung auf den RIA-Client konzipiert, sodass das Web-Frontend von SLC-β aus *technischer Sicht* für diese Arbeit von untergeordneter Bedeutung ist. Dennoch soll es der Vollständigkeit halber – abermals mit einem Schwerpunkt auf der Thematik der Barrierefreiheit – an dieser Stelle kurz vorgestellt werden.

7.1.3.1 Verwendung des Web-Frameworks *Wicket* in SLC-β

Zur Generierung der auszugebenden Seiten kommt serverseitig das Web-Entwicklungs-Framework Wicket[4] zum Einsatz. Es beschreitet einen im Vergleich zu anderen Web-Entwicklungsmethoden (wie beispielsweise JSP) neuen Ansatz insbesondere bei der Trennung der Applikationslogik von der Präsentation sowie bei der Zustandsverwaltung. Ein Entwickler von Wicket beschreibt das Framework wie folgt: "*Wicket bridges the impedance mismatch between the stateless HTTP and stateful server-side programming in Java.*"[DH09].

Wicket ist ein Framework, das in vielerlei Hinsicht zu den in dieser Arbeit betrachteten Aspekten passt, so beispielsweise auch zum Thema Barrierefreiheit. So hält Wicket beispielsweise "von Hause aus" einen Fallback-Mechanismus bereit, um für die Stellen, an denen es JavaScript verwendet, auch gleich eine scriptfreie Lösung anbieten zu können: "*Wickets fallback Ajax components provide the means to cater to users with and without JavaScript using a single code base.*"[DH09]. Insoweit setzt Wicket die in den Abschnitten 5.4.1.2ff. vorgestellten Paradigmen des *Graceful Degradation* und des *Progressive Enhancement* um.

Eine nähere Kenntnis von Wicket ist für diese Arbeit nicht weiter von Bedeutung.

Abbildung 7.1 zeigt den Begrüßungsbildschirm (nach erfolgter Anmeldung des Benutzers) und lässt erkennen, dass SLC-β auch im Textbrowser – und gemäß der Diskussionen im ersten Teil dieser Arbeit somit auch bei Verwendung eines Screenreaders – zu verwenden ist.

Aus den vorstehenden Erläuterungen zum Web-Frontend ist erkennbar, dass dieses nun nicht mehr wie in der früheren Version als Portal konzipiert wurde. Die im Vergleich zum Jahr 2000 (zu dieser Zeit entstand die erste Version von SLC) sehr stark weiterentwickelte Portal-Technologie ist auf Anwender- und Funktionsanzahl ausgerichtet, die deutlich höher liegen, als es für den Betrieb von SLC-β realistisch wäre. Die für die Fakultät – auch mittelfristig – zu erwartenden Einsatzszenarien ließen die Verwendung einer Portaltechnologie als unnötigen *Overhead* erscheinen: einem sehr hohen Aufwand stünde zu wenig Nutzen gegenüber. Vielmehr konnte das Gewünschte auch ohne Einsatz eines Portalframeworks erreicht werden.

[4] Projekthomepage: `http://wicket.apache.org`

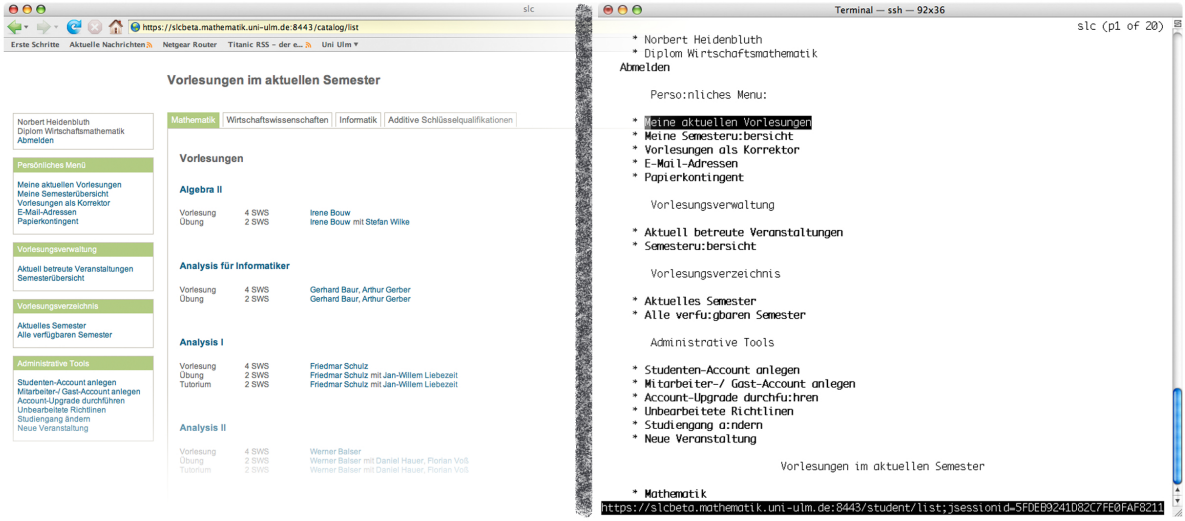


Abbildung 7.1: Begrüßungsbildschirm von SLC-β (Web-Version) in Grafik- und Textbrowsern

Dennoch war die Beschäftigung mit Portal-Frameworks und aktuellen Entwicklungen in diesem Bereich für die Entwicklung von SLC und damit auch für diese Arbeit von Bedeutung. Die Ergebnisse aus dieser Beschäftigung werden im nachfolgenden Abschnitt zusammengefasst.

7.1.3.2 Integration von RIAs und standardkonformen Portalen

Bevor die Entscheidung gegen die Verwendung einer Portaltechnik gefallen war, wurden im Rahmen der Konzeption von SLC-β mehrere Portal-Lösungen evaluiert. Sie alle erfüllten die Spezifikation JSR-168 (*"Portlet Specification"*)[5].

Eine in diesem Zusammenhang verfolgte Idee war, anstelle einer reinen Stand-Alone-RIA-Version einzelne Portlets zu integrieren, welche jeweils reichhaltige Funktionen anbieten würden. Jedes dieser reichhaltigen Portlets wäre als eigenständige Miniatur-RIA aufgetreten und hätte sich als ein Portlet in das Gesamt-Portal eingefügt.

Neben dem bereits erwähnten Portlet-Standard JSR-168 lag zeitgleich bereits eine Vorab-Version des Standards JSR-286 (*"Portlet Specification 2.0"*)[6] vor, die zu Beginn des Jahres 2008 dann endgültig verabschiedet wurde. Dieser neue Standard stellt hohe Anforderungen an die Inter-Portlet-Kommunikation, also die Kommunikation aller in einem Portal aggregierten Portlets untereinander. Dabei wird der Begriff "Portal" in beiden Spezifikationen durch leichte Nuancen in der Formulierung des ersten Satzes unterschiedlich definiert:

Definition nach JSR-168: "Ein Portlet ist eine auf Java-Technologie basierende, durch einen *Portlet Container* verwaltete Web-Komponente, die Anfragen verarbeitet und dynamischen Inhalt erzeugt. Portlets werden von Portalen als Benutzerschnittstellen verwendet, können darin beliebig hinzugefügt und entfernt werden (als sogenannte *Plug-ins*) und bilden die Präsentationsschicht für Informationssysteme."

[5] `http://jcp.org/en/jsr/detail?id=168`
[6] `http://jcp.org/en/jsr/detail?id=286`

Definition nach JSR-286: "Ein Portlet ist eine Anwendung, die einen bestimmten Inhalt (Informationen oder einen Service) bereitstellt, der in eine Portal-Seite eingefügt wird. Es wird durch einen *Portlet Container* verwaltet, welcher Anfragen verarbeitet und dynamische Inhalte erzeugt. Portlets werden von Portalen als Benutzerschnittstellen verwendet, können darin beliebig hinzugefügt und entfernt werden (als sogenannte *Plug-ins*) und bilden die Präsentationsschicht für Informationssysteme."

(*Bemerkung*: Diese Definitionen sind eigene sinngemäße Übersetzungen der in den genannten Standards gebrauchten Formulierungen.)

Der Einsatz von RIA-Portlets, welche ihrer Natur nach auf einem völlig anderen technischen Fundament stehen als traditionelle Portlets, ist in diesem Standard nicht vorgesehen. Aus diesem Grund wurde von Thomas Lange im Rahmen seiner Diplomarbeit eine Evaluation unternommen, inwieweit sich RIA-Portlets standardkonform in Portale integrieren lassen (siehe [Lan08]). Es stellte sich dabei heraus, dass "eine RIA [...] nicht aus eigener Kraft auf die Portletinformationen zugreifen [kann]"[Lan08]. Lange führt weiter aus: "Die Schwierigkeit besteht im Kommunikationsweg von der RIA zum Portlet. Eine RIA muss in der Lage sein, die Methoden eines Portlets aufzurufen. Zudem gibt es von einem Portlet nur eine einzige Instanz, mit der jeder Benutzer arbeitet. Die vorhandenen Mechanismen, um einen Benutzer und eine Portletausprägung eindeutig zu identifizieren, sind noch unzureichend."

Lange beschreibt in seiner Arbeit Lösungsansätze, kommt dabei jedoch zu dem Schluss, dass diese immer für eine bestimmte RIA-Plattform (d.h. Laufzeitumgebung) konzipiert sein müssen und es somit (derzeit) keine Universal-Lösung für die Integration von RIAs und Portalen gibt. Vor dem Hintergrund der Betrachtung eines Portlet-*Standards* – also einer Spezifikation, die von einer konkreten Implementierung abstrahiert – kann eine solche Integration momentan nicht erreicht werden. Ferner ist es aufgrund der Ergebnisse der Untersuchung nicht ratsam, ein hybrides Portal (also eines, dass sowohl RIA-Portlets als auch "normale" Portlets enthält) zu betreiben, denn dadurch – so Lange – "werden Vorteile [von RIAs] zu Nachteilen, die den Arbeitsfluss ausbremsen und stören."

Zwar hätten die Vorschläge aus [Lan08] die Entwicklung des Web-Frontend von SLC-β als Portal mit einzelnen RIA-Portlets ermöglicht. Im Hinblick auf mittelfristige Weiterentwicklungen (Erweiterungen von SLC in Richtung e-Learning-Plattform) erschien diese Möglichkeit jedoch als zu risikobehaftet und als zu stark einschränkend. Aus diesem Grunde erfolgte die nun bestehende strikte Trennung zwischen der schlank gehaltenen Web-Frontend-Version sowie der für multimediale (d.h. e-Learning-) Einsatzzwecke vorbereiteten eigenständigen RIA-Variante von SLC-β.

7.1.4 SLC on AIR: Die RIA-Version von SLC-β

Die RIA-Version von SLC-β ist dazu konzipiert, ein möglichst reichhaltiges Benutzererlebnis anzubieten. Wie eingangs bereits erwähnt, bedeutet dies konkret, dass multimediale und interaktive Elemente, die sich nicht oder nur eingeschränkt als Web-Version anbieten lassen, über die RIA-Version zur Verfügung gestellt werden können, wobei hier insbesondere der Einsatz als e-Learning-Plattform adressiert wird. Ebenso spielte auch der Wunsch der ubiquitären Verfügbarkeit von SLC eine wichtige Rolle bei der Konzeption: das System soll nicht nur online via Computer verwendet werden können, sondern auch einen Offline-Betrieb erlauben sowie – idealerweise – auf mobilen Endgeräten lauffähig sein.

Wie zuvor bereits angedeutet, wurde bei der Konzeption des RIA-Clients zunächst die Gruppe der Studierenden als Benutzerklientel identifiziert. Während der Funktionsumfang des kompletten SLC-Systems nicht nur studentische Belange abbildet, sondern auch Funktionen für administrative Aufgaben wie Vorlesungsplanung und Benutzerverwaltung anbietet, wurden letztere als für einen RIA-Client von untergeordneter Bedeutung bewertet. Der RIA-Client sollte nicht als Selbstzweck dienen und mit einer Fülle von weitgehend nicht verwendeten Funktionen aufwarten, sondern einen Teilbereich aus denjenigen Funktionen des Backends anbieten, die eine sinnvolle und eine an *User Experience* (vgl. Seite 89) Gewinn bringende Umsetzung als *Rich Internet Application* erlauben. Zugleich soll der Umgang mit der RIA-Version (Installation, Updates, Deinstallation) möglichst einfach sein.

Mit diesen Vorgaben wurden die in Kapitel 4.2 vorgestellten RIA-Technologien auf ihre Eignung hin evaluiert. Unter den Alternativen hat sich zum Zeitpunkt der Evaluation die Adobe-AIR-Technologie von Adobe (siehe dazu Abschnitt 4.2.2 auf Seite 85) als in allen erforderlichen Punkten überlegen herausgestellt. Auch der Wunsch, die Software auf einem möglichst breiten Spektrum an Geräten lauffähig zu haben, beeinflusste die Entscheidung pro Adobe AIR maßgeblich. Somit fiel die Entscheidung, den RIA-Client von SLC Beta für die AIR-Plattform zu entwickeln, und hieraus ergab sich auch der fortan verwendete Projektname des RIA-Clients: “SLC on AIR”.

An dieser Stelle seien noch einmal die wesentlichen Vorteile, welche die AIR-Technologie bietet, aufgeführt:

- Unter allen (zum Zeitpunkt der Evaluation) am Markt vorhandenen Alternativen ist AIR mit großem Abstand technologisch am weitesten entwickelt. AIR basiert auf der Flash-Plattform, die mit über 10 Jahren Marktpräsenz eine verlässliche Stabilität und Reife aufweist.

- AIR sowie das zugrunde liegende Framework Flex sind kostenlos erhältlich und verwendbar.

- AIR-Applikationen laufen auf allen gängigen Betriebssystemen (Windows, Mac, Linux).

- Das AIR-Framework ist dafür konzipiert, auch auf mobilen Endgeräten einsatzfähig zu sein. Für die zweite Jahreshälfte 2009 sind mobile Endgeräte zu erwarten, die leistungsfähig genug sind, um AIR-Applikationen zu unterstützen[7]. (Zum Zeitpunkt der Fertigstellung dieser Arbeit im November 2009 ist diese Ankündigung aber noch nicht Realität geworden.)

- Das AIR-Framework enthält eine SQLite-Datenbank, mit deren Hilfe auch ein Offline-Betrieb des SLC-RIA-Clients möglich wird: die relevanten Daten werden lokal abgelegt, können offline verarbeitet werden und erfahren beim nächsten Verbinden mit dem SLC-Backend eine Synchronisation.

- Die multimedialen Fähigkeiten von Flash/AIR sind sehr weit ausgereift, was für den Einsatz als e-Learning-Technologie ein sehr wichtiges Entscheidungskriterium darstellt.

- AIR unterstützt nativ die Darstellung von HTML- und PDF-Dokumenten. (Dies ist günstig für den Einsatz in einem e-Learning-Szenario, weil so die Betrachtung von Skripten, Übungsblättern und weiteren vorlesungsbegleitenden Materialien einfach in die Benutzerführung integriert werden können.)

[7] siehe: `http://www.arm.com/news/23776.html`

- Im Unterschied zu browserbasierten Anwendungen (auch Flash) erlaubt AIR den Zugriff auf das lokale Dateisystem. Somit lässt sich der Up- und Download von Dateien (Skripte, Übungsaufgaben etc.) sehr einfach realisieren. Durch die ebenfalls vorhandene Unterstützung von Drag-'n'-Drop kann hier für die Studierenden eine sehr einfache Verwaltung ihrer vorlesungsspezifischen Dateien angeboten werden.

- In Kombination mit dem bereits in einem früheren Kapitel verwendeten OpenSource-Produkt "BlazeDS" lassen sich Remoting- und Messaging-Funktionalitäten in AIR realisieren. Somit kann eine Client-zu-Client-Kommunikation durchgeführt werden (z.B. ein Chatraum für einzelne Veranstaltungen), aber auch der Zugriff auf Java-Klassen des Backends aus der AIR-Anwendung heraus ist möglich (Remoting). Dies macht den Einsatz von Web-Services obsolet.

- Wie im Abschnitt über das Backend von SLC-β beschrieben (siehe Abschnitt 7.1.2), wurde dies unter Verwendung des Spring Frameworks entwickelt. Im Frühjahr 2009 wurde die "Spring BlazeDS Integration" veröffentlicht (siehe [Spr09]), die speziell dabei helfen soll, RIAs mit Adobe Flex zu entwickeln, die auf ein Spring-basiertes Backend-System zugreifen. Somit ergibt sich ein nahtloser Übergang zwischen Backend und RIA-Client, der in dieser Form mit alternativen RIA-Frameworks (zum Zeitpunkt der Entstehung dieser Arbeit) nicht möglich wäre.

- AIR-Applikationen lassen sich einfach und ohne administrative Eingriffe auf dem Zielrechner installieren und ebenso leicht auch wieder deinstallieren. Desweiteren ist in AIR ein Update-Framework enthalten, das ein Einspielen von Updates von AIR-Applikationen sehr komfortabel macht.



Abbildung 7.2: Screenshot des Begrüßungsbildschirms von SLC on AIR

Einen optischen Eindruck des RIA-Clients vermittelt Abbildung 7.2. Vergleicht man den darin gezeigten Begrüßungs-Screenshot mit jenem der Web-Version (Abbildung 7.1 auf Seite 185,

so wird unmittelbar erkennbar, dass hier – neben den reichhaltigen Möglichkeiten für einen späteren Einsatz als e-Learning-Plattform – auch die emotionale Ansprache der Software an den Benutzer (vgl. die Ausführungen über *User Experience* in Abschnitt 4.3.2 auf Seite 89) eine tragende Rolle bei der Entwicklung der RIA-Version spielte.

7.1.4.1 Bedeutung des RIA-Clients für diese Arbeit

Letztendlich werden mit fortschreitender Entwicklung der Alternativen zu Adobe AIR (vgl. dazu Abschnitte 4.2.3ff.) auch jene die genannten Vorteile besitzen. Insofern soll für den weiteren Verlauf dieser Arbeit die Tatsache, dass die Entscheidung für eine bestimmte Technologie gefallen ist, in den Hintergrund treten.

In den folgenden Abschnitten wird die Anwendung der Pattern aus Kapitel 6 auf den RIA-Client näher erläutert. Er stellt in dieser Arbeit die Referenz-Implementierung der aufgezeigten Pattern dar und zitiert deshalb die Programmtexte erneut in der Sprache *ActionScript* bzw. "MXML". Grundsätzlich sind die Pattern jedoch unabhängig von einer bestimmten RIA-Plattform und insoweit prinzipiell auf alle derartige Plattformen anwendbar.

7.2 Anwendung der Pattern auf SLC on AIR

Die in Kapitel 6 beschriebenen Problemstellungen sowie die dort vorgestellten Lösungsmöglichkeiten sind unmittelbar in die Entwicklung von SLC on AIR eingegangen. In diesem Abschnitt sollen sie nun nochmals aufgegriffen und ihre Anwendung auf SLC on AIR erläutert werden.

7.2.1 Das SSC-Pattern in SLC on AIR

Es wurde bereits an verschiedenen Stellen darauf hingewiesen, dass der RIA-Client von SLC-β auf die Zielgruppe "Studierende" fokussiert. Zugleich war auch der Wunsch, sowohl online als auch – eingeschränkt – offline arbeiten zu können, eine an den Client gestellte Anforderung. Allein aus diesen Tatsachen lassen sich unmittelbar drei Zustände ableiten, in denen die Applikation betrieben werden kann:

1. Der Benutzer ist online und hat sich am Backend authentifiziert (angemeldet).
2. Der Benutzer ist online, hat aber auf eine Authentifizierung verzichtet oder sich bereits wieder vom Backend abgemeldet.
3. Der Benutzer arbeitet offline. Eine Authentifizierung am Backend ist somit nicht möglich.

Der Funktionsumfang von SLC on AIR beinhaltet Funktionen, die eine bestehende Anmeldung zwingend erfordern, desweiteren gibt es Funktionen, die zwar eine Online-Verbindung zum Backend voraussetzen, nicht jedoch eine Anmeldung. Und letztlich gibt es auch solche Funktionen, die offline betrieben werden können. Abbildung 7.3 visualisiert dies.

In der Nomenklatur des in Abschnitt 6.3 vorgestellten SSC-Patterns bedeutet dies, dass die einzelnen Funktionen von SLC on AIR *sensitiv* für einerseits den Login-Status sowie andererseits den Konnektivitäts-Status sein müssen. Beide Zustände können sich zur Laufzeit jederzeit

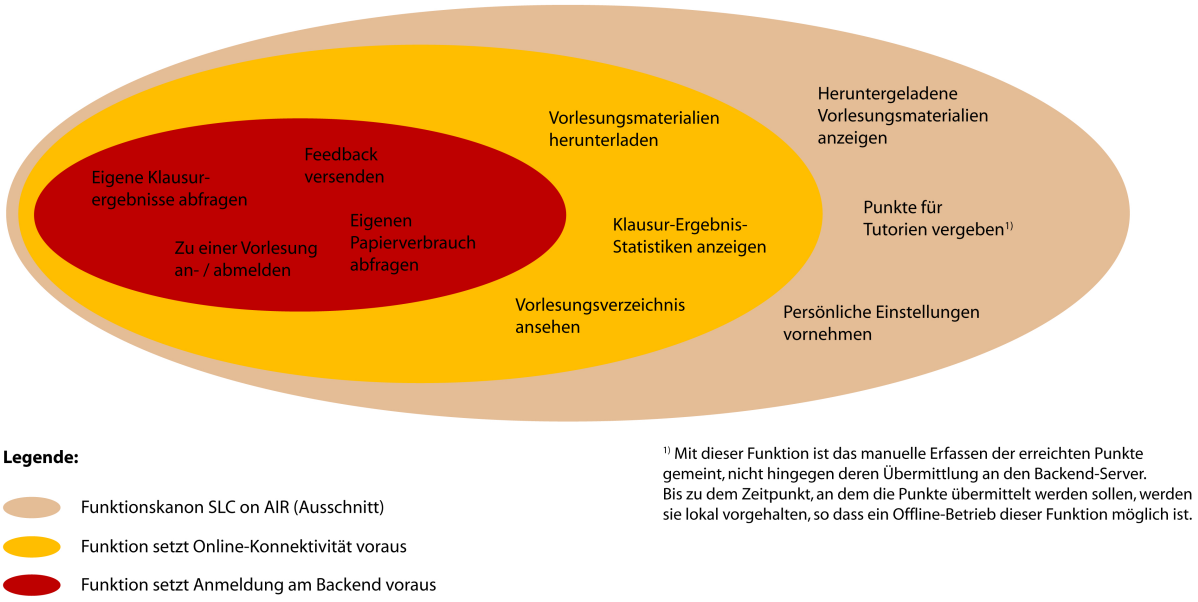


Abbildung 7.3: Funktionskanon und Status-Voraussetzungen in SLC on AIR

ändern, wobei zu beachten ist, dass aus dem Vorliegen des Zustands “Benutzer ist offline” unmittelbar auch folgen muss, dass der Benutzer nicht mehr angemeldet ist. (Die Rückrichtung dieser Aussage gilt indes nicht!)

Jedoch liegt SLC on AIR als in sich geschlossene Applikation auf dem Client-Rechner vor, und damit birgt es den gesamten Funktionskanon in sich. Es darf aber nur die Teilmenge an Funktionen präsentieren, die zum aktuellen Kontext[8] passt.

Ein Abbruch der Online-Verbindung muss also dazu führen, dass der angebotene Funktionsumfang sofort an den neuen Kontext angepasst wird. Analog verhält es sich bei der Änderung des Login-Status’. Diese Anpassung muss sich im Falle von SLC on AIR auf zwei Bereiche des Benutzerinterfaces auswirken: auf die Navigationsstruktur und auf den Hauptbereich. Diese Bereiche sind in der noch folgenden Abbildung 7.5 (siehe Seite 193) mit “1” bzw. “2” gekennzeichnet.

7.2.1.1 Anpassen der Navigationsstrukturen an den neuen Kontext

Um zu erreichen, dass sich das Menü von SLC on AIR (dies ist zugleich auch dessen einzige Navigationsstruktur) jederzeit an den Kontext anpassen kann, wurde die für die Repräsentierung des Menüs zuständige GUI-Komponente um die Implementierung der Interfaces `ILoginSensitive` und `IOnlineSensitive` erweitert und mithin zu einer zustandssensitiven Komponente gemacht. Gemäß dem SSC-Pattern obliegt es ja jeder Komponente selbst, auf die Änderungen im Kontext geeignet zu reagieren, und im Fall von SLC on AIR ist das Menü (repräsentiert durch die GUI-Komponente `MenuBar`) die (einzige) Komponente, die auf Kontextänderungen reagiert. Ziel der Anpassung der Komponente `MenuBar` ist es, dass sie ihre Einträge je nach Kontext aktiviert oder deaktiviert. Visuell kommt ein deaktivierter Menüeintrag dadurch zum Ausdruck, dass er in einer anderen Farbe (grau) dargestellt wird und vor allem nicht ausgewählt werden kann. Das bedeutet, dass deaktivierte Einträge zwar grundsätz-

[8] Der Begriff Kontext ist hier gemäß der Definition auf Seite 162 zu verstehen.

lich in den Menüstrukturen aufgeführt (und somit sichtbar) sind, die ihnen hinterlegten Funktionen aber nicht aufgerufen werden können.

Flex bzw. die damit verwendete Programmiersprache *ActionScript* nebst der deklarativen GUI-Sprache MXML machen den Umgang mit Menüstrukturen sehr einfach und ermöglichen auch eine gute Integration mit dem SSC-Pattern. Die eigentliche Menüstruktur liegt als XML-Dokument vor. Listing 7.1 zeigt ein Beispiel für ein solches XML-Dokument, welches in diesem Fall das deutschsprachige Menü repräsentiert. Dem Listing ist zu entnehmen, dass ein Menü aus einer hierarchischen Kombination von XML-Tags namens `menuitem` besteht. Diese *MenuItems* können demnach sowohl als Endpunkt als auch als Vaterknoten für weitere *MenuItems* verwendet werden und erlauben somit den Aufbau beliebig komplexer Menüstrukturen.

Listing 7.1: XML-Menüstruktur in SLC on AIR

```
<root>
  <menuitem label="Dein Account" data="top" connSensitivity="none" enabled="true">
    <menuitem label="Papierverbrauch" data="paper" connSensitivity="login" enabled="true"/>
    <menuitem type="separator" connSensitivity="none" enabled="true"/>
    <menuitem label="Aktuelle Meldungen" data="motd" connSensitivity="online" enabled="true"/>
  </menuitem>
  <menuitem label="Vorlesungsmanagement" data="top" connSensitivity="none" enabled="true">
    <menuitem label="Vorlesungsverzeichnis" data="lecMgmt" connSensitivity="online" enabled="true"/>
    <menuitem label="Punkte abfragen" data="assResults" connSensitivity="login" enabled="true"/>
  </menuitem>
    <menuitem label="Einstellungen" data="top" connSensitivity="none" enabled="true">
      <menuitem label="Sprache" data="top" connSensitivity="none" enabled="true">
        <menuitem label="Deutsch" toggled="true" type="radio" groupName="lang" selected="true"
          data="langToGerman" connSensitivity="none" enabled="true"/>
        <menuitem label="Englisch" toggled="false" type="radio" groupName="lang"
          data="langToEnglish" connSensitivity="none" enabled="true"/>
    </menuitem>
  </menuitem>
  <menuitem label="SLC on AIR" data="top" connSensitivity="none" enabled="true">
    <menuitem label="Feedback senden" data="sendFeedback" connSensitivity="login" enabled="true"/>
    <menuitem label="Credits" data="credits" connSensitivity="none" enabled="true"/>
    <menuitem type="separator" connSensitivity="none" enabled="true"/>
    <menuitem label="Beenden" data="quitSLC" connSensitivity="none" enabled="true"/>
  </menuitem>
</root>
```

Jeder *MenuItem*-Eintrag kann eine Reihe von Eigenschaften besitzen. Eine solche Eigenschaft ist beispielsweise die Eigenschaft `label`, die angibt, unter welcher Bezeichnung der jeweilige Eintrag im Menü zur Anzeige kommt. Für den hier betrachteten Zusammenhang sind jedoch zwei weitere Eigenschaften von Bedeutung: `connSensitivity`[9] und `enabled`. Die Eigenschaft `enabled` signalisiert, ob der jeweilige Eintrag aktiviert oder deaktiviert ist und dementsprechend ausgewählt werden kann oder nicht. Die Eigenschaft `connSensitivity` wird so gesetzt, dass aus ihr hervorgeht, ob der jeweilige Menüeintrag sensitiv ist (und wenn ja: in welcher Weise). Sie kann die folgenden Werte annehmen:

online: Der Menüeintrag ist genau dann aktiviert, wenn eine Online-Verbindung besteht.

login: Der Menüeintrag ist genau dann aktiviert, wenn der Benutzer am Backend angemeldet ist. (Dies setzt zugleich eine bestehende Online-Verbindung voraus).

none: Der Menüeintrag steht immer zur Verfügung – unabhängig vom Kontext. Dies bedeutet, dass der entsprechende Eintrag *nicht* sensitiv ist.

Die zustandssensitive GUI-Komponente `MenuBar`, welche (in Flex) die optische Repräsentierung einer Menüstruktur der vorgestellten Art übernimmt, kann nun, wie in Abbildung 7.4 skizziert, auf eine Statusänderung reagieren: für jeden Menü-Eintrag wird geprüft, ob dessen Eigenschaft `connSensitivity` zum aktuellen Kontext passt. Davon abhängig wird dann die Eigenschaft `enabled` gesetzt.

[9] eine Kurzform für "*Connection Sensitivity*"

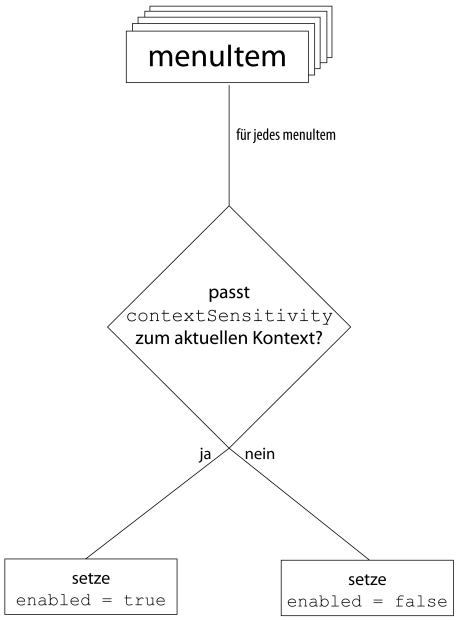


Abbildung 7.4: Algorithmus zum Anpassen der Menüstruktur an einen neuen Kontext (Skizze)

Die nachfolgende Codezeile (Listing 7.2) zeigt, dass in Flex aufgrund dessen Unterstützung des ECMA-Standards (siehe [ECM99]) bzw. von E4X[10] der skizzierte Algorithmus *mit nur einer Anweisung* umgesetzt werden kann. Im Beispiel werden alle diejenigen Menüeinträge deaktiviert, die einen bestehenden Login voraussetzen.

Listing 7.2: Deaktivierung von login-sensitiven Menüeinträgen

```
1 menuStructure..menuitem.(@connSensitivity=="login").(@enabled=false);
```

Erläuterung zu Listing 7.2: Das Objekt `menuStructure` sei hier die in Listing 7.1 gezeigte XML-Struktur. Die Notation mit zwei aufeinanderfolgenden Punkten gibt an, dass jedes Kind-Tag `menuitem` der Menüstruktur (unabhängig von dessen Position in der Menü-Hierarchie) betrachtet werden soll. Aus der Menge aller dieser Kind-Tags werden durch die Notation `(@connSensitivity=="login")` diejenigen herausgefiltert, deren Eigenschaft `connSensitivity` den Wert `login` besitzt, mithin also alle login-sensitiven Einträge. Der letzte Bestandteil des Ausdrucks setzt schließlich die Eigenschaft `enabled` der herausgefilterten Menüeinträge auf `false`.

Die GUI-Komponente `MenuBar` kann folglich mit minimalem Aufwand zu einer zustandssensitiven Komponente erweitert werden, die nur solche Menüeinträge aktiviert, die dem aktuellen Kontext entsprechen, und alle anderen deaktiviert (aber dennoch anzeigt). Ein anderer Ansatz, die Menüstrukturen an den Kontext anzupassen, besteht darin, sie aus der Menüstruktur zu entfernen. Dadurch sind sie gar nicht mehr sichtbar. Diese Idee wird ebenfalls im Rahmen dieser Arbeit besprochen (in Zusammenhang mit der Adaption von Navigationsstrukturen an Benutzerprivilegien, vgl. Abschnitt 7.2.2 auf Seite 196).

Abbildung 7.5 zeigt den Aufbau des Benutzer-Interfaces von SLC on AIR in Form eines

[10] Dies ist die Abkürzung für "ECMAScript für XML", einer im ECMA-Standard 357 spezifizierten XML-Unterstützung in ECMA-konformen Programmiersprachen.

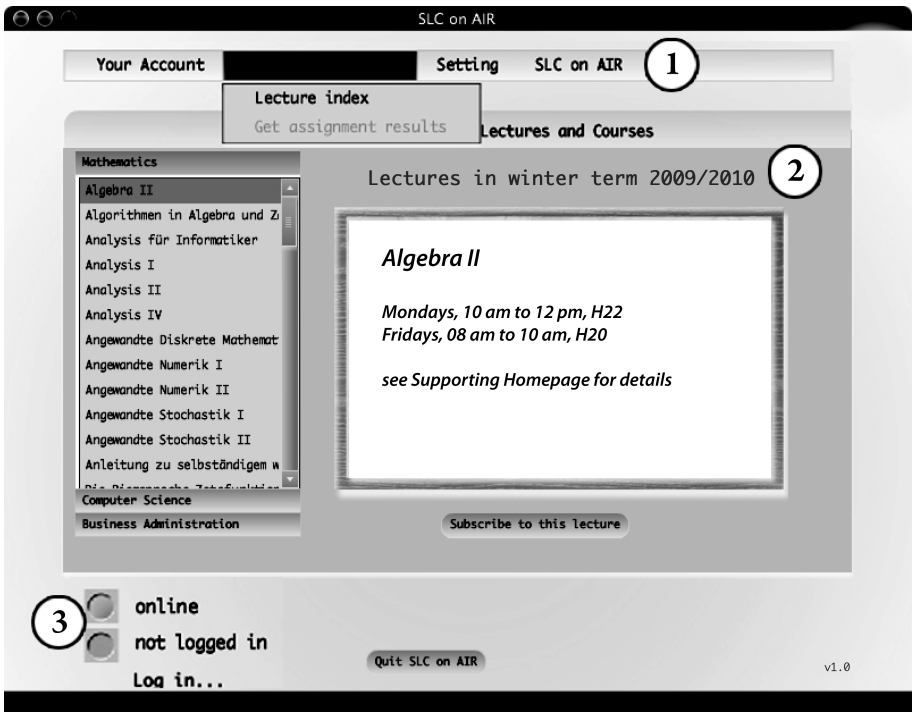


Abbildung 7.5: Struktur der GUI in SLC on AIR

Screenshots[11]. Die drei wesentlichen Bestandteile des Interfaces sind mit den Ziffern 1 bis 3 bezeichnet und besitzen die folgende Bedeutung:

Menüleiste (1): Die in diesem Abschnitt erläuterte Menüleiste ist in der Abbildung mit der Ziffer 1 markiert; die optische Unterscheidung zwischen aktivierten Menüeinträgen ("Lecture index") und deaktivierten ("Get assignment results") ist in der Abbildung ebenfalls erkennbar.

Hauptbereich (2): Der mit der Ziffer 2 markierte Hauptbereich ("*Main Panel*") des Interfaces stellt jeweils die GUI-Komponenten für eine bestimmte Funktionalität zur Verfügung (vgl. nächster Abschnitt). In der Abbildung ist dies das Vorlesungsverzeichnis für Vorlesungen im Wintersemester 2009/2010.

Statusindikatoren (3): SLC on AIR überwacht zwei Zustände: den Online-Status und den Login-Status. Der Benutzer ist durch die mit der Ziffer 3 markierten Statusindikatoren ebenfalls zu jeder Zeit darüber informiert, ob er an- oder abgemeldet und die Anwendung online oder offline ist.

7.2.1.2 Deaktivieren des aktuellen *Main Panels*

Neben der Menüstruktur (1) ist auch der Hauptbereich (in Abbildung 7.5 mit 2 markiert und im Folgenden als *Main Panel* bezeichnet) als zustandssensitive Komponente realisiert (hier: als Erweiterung der GUI-Container-Klasse `Canvas`). Auf diese Weise kann sichergestellt werden, dass Kontextänderungen auch in der aktuell verwendeten Funktionalität[12] berücksichtigt

[11] Der Screenshot wurde zur besseren Darstellung im Druck nachbearbeitet.

[12] Diese Begriff sei hier als Synonym für den Terminus "funktionale Einheit", der auf Seite 171 definiert wurde, verwendet.

werden.

Würde der Hauptbereich nicht als zustandssensitive Komponente vorliegen, so führte dies dazu, dass mit einer Änderung des Kontexts zwar die Menüstruktur reagiert, aber die im Hauptbereich angezeigte Funktionalität noch weiterhin verwendbar bliebe. Zwar lassen sich andere Funktionen im nun angepassten Menü nicht mehr auswählen, aber die zuletzt gewählte Funktion wird noch immer im Hauptbereich dargestellt, auch wenn dies aufgrund der Änderungen im Kontext gar nicht mehr sein darf.

Beispiel: Ein Benutzer befindet sich aktuell in der Funktion "Feedback schreiben". Diese Funktion setzt eine bestehende Anmeldung am Backend voraus. Meldet sich der Benutzer nun ab, so darf das Versenden eines Feedbacks fortan nicht mehr möglich sein. Die entsprechenden Steuerelemente der GUI – insbesondere die Schaltfläche "Absenden" – müssen sofort deaktiviert werden.

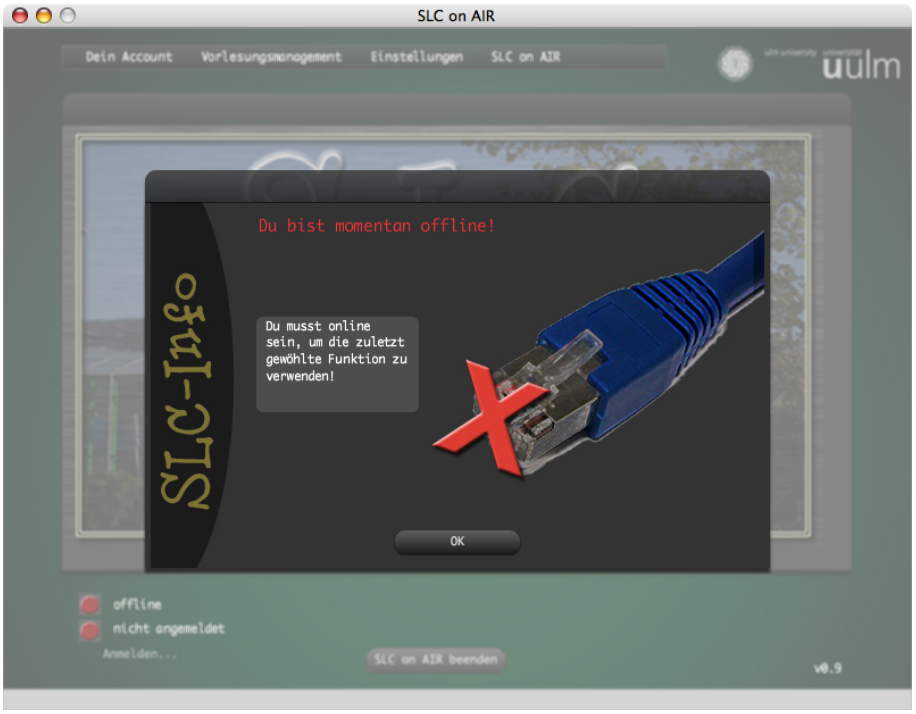


Abbildung 7.6: Reaktion einer online-sensitiven Komponente auf den Abbruch der Online-Verbindung

Mit Hilfe des SSC-Patterns kann das Gewünschte erreicht werden: wenn der im Hauptbereich zur Anzeige zu bringende GUI-Container `MainPanel` als zustandssensitive Komponente implementiert (bzw. zu einer solchen erweitert) wird, dann wird auch er auf die Kontextänderungen reagieren können. Im Falle von SLC on AIR rufen derartige Komponenten bei Bedarf eine Methode auf, die dazu führt, dass der Benutzer zunächst über das Nicht-Bestehen des erforderlichen Kontexts informiert und sodann zurück auf den Begrüßungsbildschirm von SLC on AIR geführt wird. Abbildung 7.6 zeigt dies am Beispiel: hier wurde eine als online-sensitiv markierte Komponente im Hauptbereich angezeigt, die Konnektivität ging jedoch verloren. Die Abbildung zeigt die Reaktion von SLC on AIR auf diesen Umstand: zunächst wird der Benutzer per PopUp-Fenster informiert, und nach dem Bestätigen dieses Fensters (durch Klicken auf OK) wird er zurück in den Begrüßungsbildschirm geleitet. Damit steht ihm zeitgleich

mit der Veränderung des Kontexts (hier: mit dem Abbruch der Konnektivität) die zuletzt verwendete Funktion nicht mehr zur Verfügung, weil diese das Vorhandensein einer Online-Verbindung voraussetzt. Eine analoge Reaktion erfolgt auf einen Abmeldevorgang, wenn – wie im obigen Beispiel zur Feedback-Komponente – aktuell eine login-sensitive Komponente angezeigt wird (siehe dazu den Screenshot in Abbildung 7.7).

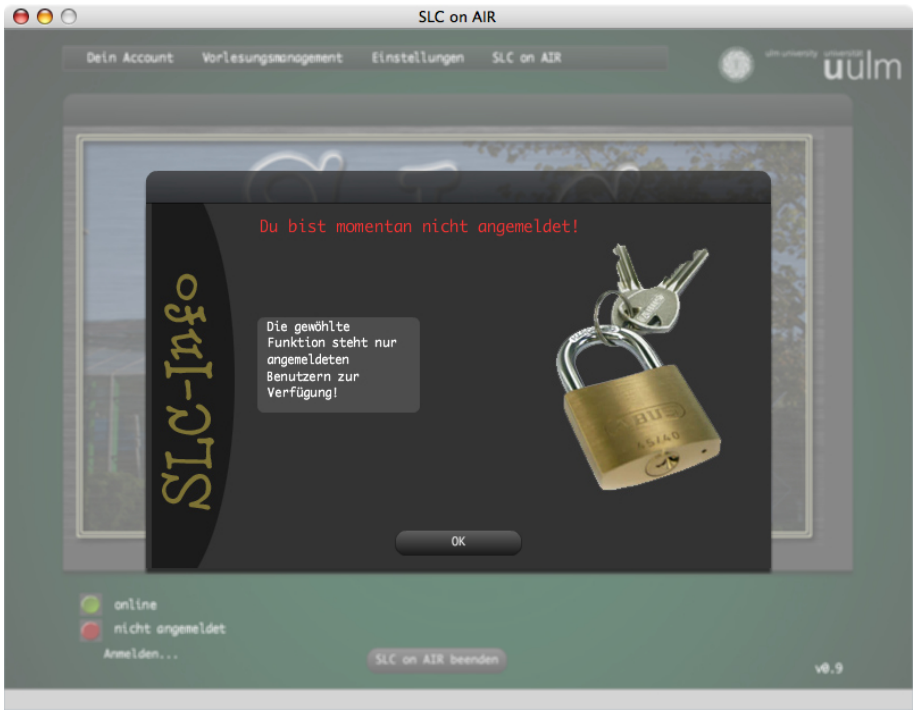


Abbildung 7.7: Reaktion einer login-sensitiven Komoponente auf einen Abmeldevorgang

7.2.1.3 Abmelden einer zustandssensitiven Komponente beim StatusChangeManager

Dem UML-Sequenzdiagramm in Abbildung 6.3 auf Seite 168 ist zu entnehmen, dass neben der Anmeldung der zustandssensitiven Komponenten beim *StatusChangeManager* auch ihr dortiges Abmelden von Bedeutung ist. Unterbleibt diese Abmeldung, so hält der *StatusChangeManager* noch eine Referenz auf diese Komponente, was ein Abräumen durch den *Garbage Collector* verhindert. In der Folge bedeutet dies, dass die Komponente zwar nicht mehr angezeigt wird, aber dennoch existiert und demzufolge bei der nächsten Änderung des Kontexts reagieren würde, was zu sehr schwer zu lokalisierenden Funktionsstörungen führen kann. Da diese Situation insbesondere dann auftreten kann, wenn sich der Hauptbereich (`MainPanel`) ändert (ein Panel wird durch einen anderen ersetzt, weil eine Funktion aus dem Menü ausgewählt wurde), wird an dieser Stelle noch ein Code-Beispiel dafür gegeben, wie eine ordnungsgemäße Abmeldung einer zustandssensititven Komponente in SLC on AIR erfolgen sollte. (siehe Listing 7.3).

Listing 7.3: Abmeldung einer zustandssensitiven Komponente vom *StatusChangeManager*

```
1 private function removePanel() : void {
2    var p: StatusSensitiveComponent =
3       (mainPanel.getChildAt(0)) as StatusSensitiveComponent;
4    p.unregister();
```

```
5     mainPanel.removeChild(p);
6 }
```

Erläuterung: Das Codebeispiel zeigt, wie sich ein Hauptbereichs-Container (`mainPanel`) beim *StatusChangeManager* abmelden kann. Bestandteil einer jeden zustandssensitiven Komponente ist die Methode `unregister()`, die durch das Interface `ISensitive` vorgeschrieben wird (vgl. Listing 6.1 auf Seite 165). Deshalb wird die im Hauptbereich angezeigte Komponente (sie ist das einzige Kind-Element des Containers `mainPanel` und kann somit durch den Aufruf von dessen Methode `getChildAt(0)` erreicht werden) veranlasst, ihre Methode `unregister` aufzurufen. Wie dem UML-Sequenzdiagramm (Abbildung 6.3 auf Seite 168) zu entnehmen ist, führt diese Methode lediglich zum Aufruf von `unsubscribe()` auf dem *StatusChangeManager*, welcher dann – je nach zugrundeliegendem Benachrichtigungsmechanismus – die geeignete Maßnahme trifft, um die entsprechende Komponente zukünftig *nicht* mehr zu benachrichtigen. Somit wird an dieser Stelle also eine bestehende Referenz aufgelöst. Die letzte Zeile im Codebeispiel entfernt die Komponente aus dem Hauptbereich, sodass ab diesem Moment keine weitere Referenz mehr auf die zu entfernende Komponente existiert. Beim nächsten Durchlauf des *Garbage-Collectors* wird sie daher abgeräumt und kann fortan nicht mehr zu Seiteneffekten bei der Benachrichtigung über Kontextänderungen führen.

7.2.2 Privilege Aware Menus

Unabhängig davon, ob der RIA-Client oder die Web-Version verwendet wird, unterscheidet das SLC-β-System Benutzer nach den ihnen zugeteilten Berechtigungen. In der zum Zeitpunkt der Entstehung dieser Arbeit aktuellen Version finden die folgenden Berechtigungen Verwendung:

Account: berechtigt zum Anlegen von neuen Studenten-Accounts

Admin: berechtigt zum administrativen Systemzugang

Ladmin: berechtigt zum Anlegen und Editieren von Vorlesungen

Lecturer: Standard-Berechtigung für Dozenten

Student: Standard-Berechtigung für Studierende

TA: erweiterte Berechtigungen für Studierende, die als wissenschaftliche Hilfskräfte (*Teaching Assistants*) mitarbeiten

Während die Web-Version in ihrer Eigenschaft als klassische Client-Server-Applikation den Umgang mit Berechtigungen auf traditionelle Weise handhabt, stellten die Berechtigungen an die RIA-Variante neue Anforderungen: prinzipiell sollte die RIA-Version alle Funktionen[13] enthalten, aber nur dann auch zur Benutzung anbieten, wenn dies adäquat zu den bestehenden Berechtigungen des aktuellen Benutzers ist.

Aus dieser Problemstellung heraus entstand das in Abschnitt 6.4 beschriebene Konzept der *Privilege Aware Menus* ("PAM-Pattern"). Es erlaubt im RIA-Client von SLC, dass jeder angemeldete Benutzer genau die Teilmenge aller verfügbaren Funktionen im Menü sieht, die seinen

[13] Bei dieser Gelegenheit sei nochmals erwähnt, dass SLC on AIR Studierende als Benutzer adressiert und insofern mit "alle Funktionen" gemeint ist: "alle für Studierende relevante Funktionen".

individuellen Berechtigungen entspricht. In der in Abschnitt 6.4 beschriebenen Weise sind die Funktionen als funktionale Einheiten organisiert (vgl. Definition auf Seite 171), d.h. sie sind vollumfänglich in (nur) einer Klasse enthalten, welche im konkreten Fall von der Klasse `Canvas` abgeleitet wurde, um sich so in das *MainPanel* (Bereich "2" in Abbildung 7.5 auf Seite 193) einzubringen. Objekte dieser Klasse sind in Flex Erweiterungen einfacher GUI-Container, wie sie (zum Teil unter anderer Bezeichnung) in allen gängigen GUI-Frameworks zu finden sind. Zugleich wurden sie um die Implementierung des Interfaces `PrivilegeAwareInterface` ergänzt und erfüllen damit die Grundvoraussetzung für die Anwendung des PAM-Patterns.

Bei Programmstart und bei jeder Änderung am Login-Status wird der in Abbildung 6.4 auf Seite 172 skizzierte Prozess durchlaufen: jede funktionale Einheit wird "kontaktiert" und um Meldung der benötigten Berechtigungen gebeten. Je nachdem, was der anschließende Vergleich mit den aktuell vorliegenden Berechtigungen ergibt, wird der zugehörige Eintrag entweder aus der Navigationsstruktur entfernt oder verbleibt in dieser.

7.2.3 Multilinguale Komponenten

Im Hinblick auf die internationalen Studierenden der Universität Ulm ist die Unterstützung von Mehrsprachigkeit in SLC on AIR von gewichtiger Bedeutung. Mit dem MLL-Pattern steht eine bequeme Möglichkeit bereit, SLC on AIR in vielen verschiedenen Sprachversionen verfügbar zu haben, sodass SLC on AIR Gebrauch von diesem Pattern macht.

Aktuell unterstützt SLC on AIR die Sprachen Deutsch und Englisch, jedoch lassen sich aufgrund der Ausführungen in Abschnitt 6.5 auf Seite 177 sehr einfach Ergänzungen der verfügbaren Sprachen realisieren. Der Umstand, dass aktuell nur zwei Sprachen unterstützt werden, ist demzufolge nicht technischer Natur, sondern durch insuffiziente Kenntnisse weiterer Sprachen bedingt. Es wurde aus diesem Grunde bereits überlegt, fremdsprachige Studierende um die notwendigen Übersetzungshilfen zu bitten, um auf diese Weise eine möglichst umfangreiche Sprachunterstützung des SLC-Systems anbieten zu können. Weil SLC on AIR (bzw. das zugrunde liegende Flex) die UTF-8-Codierung verwendet, ließe sich ebenfalls eine Übersetzung in Sprachen realisieren, die nicht auf dem lateinischen Zeichensatz basieren (beispielsweise in einen chinesischen Dialekt). Da die Übersetzungsarbeiten im Wesentlichen organisatorische, nicht jedoch technische Probleme erwarten lassen, wurde auf die Übertragung von SLC on AIR in weitere Sprachen im Rahmen dieser Arbeit verzichtet.

7.2.4 Versionskontrolle

Wann immer plug-in-basierte RIAs oder traditionelle Web-Applikationen verwendet werden, liefert sie ein Web-Server neu aus, sodass sie stets aktuell präsentiert werden können. Sobald eine aktualisierte Version (ein *Update*) der RIA bzw. der Web-Anwendung erstellt und auf dem Server veröffentlicht wird, erhält der Benutzer ab dem nächsten Zugriff die neueste Fassung der Anwendung.

Bei Stand-Alone-RIAs ist ein solches Vorgehen aber nicht möglich, da die RIA als lokale Applikation auf dem Rechner des Benutzers vorliegt und sich damit außerhalb der Zugriffssphäre des Web-Servers befindet. Dies gilt zwar bei traditionellen Desktop-Anwendungen ebenso, jedoch ist RIAs – im Vergleich zu Desktop-Anwendungen – ein eher volatiler und somit schneller alternder Charakter zu bescheinigen. Aus diesem Grund ist es sinnvoll, einen Mechanismus zu verwenden, der eine veraltete RIA-Version "aus der Ferne" deaktivieren kann. Ein solcher Mechanismus wurde in SLC on AIR implementiert. Er ist insbesondere – und aus diesem

Grunde soll er an dieser Stelle auch Erwähnung finden – ein weiteres Anwendungsbeispiel für das SSC-Pattern.

7.2.4.1 Serverseitiges Vorhalten der aktuellen Version

Im SLC-Backend wird die Versionsnummer des aktuellsten RIA-Clients hinterlegt. Dies erfolgt in Form einer simplen XML-Datei, die in Listing 7.4 gezeigt ist.

Listing 7.4: Serverseitige Information zur aktuellsten RIA-Client-Version

```
1 <versionInfo>
2    <curVersion>v1.00</curVersion>
3    <upgradeReq>silent</upgradeReq>
4 </versionInfo>
```

Die Eigenschaft `upgradeReq` gibt dabei an, welche Auswirkung es haben soll, wenn die verwendete RIA-Version eines Benutzers nicht der aktuell verfügbaren Version entspricht. Dazu sind drei Möglichkeiten (Eigenschaftsausprägungen) vorgesehen:

silent: Es liegt zwar eine aktuellere Version des RIA-Clients vor, jedoch wird darauf *nicht* hingewiesen, wenn ein Benutzer eine ältere Version verwendet.

suggested: Auf dem Server liegt eine neue(re) Version vor, jedoch wird deren Verwendung nur empfohlen. Sie ist also nicht zwingend nötig.

mandatory: Die neue(re) Version ist zwingend erforderlich, um mit SLC on AIR weiterzuarbeiten. Dies kann beispielsweise aufgrund der Behebung eines kritischen Sicherheitsproblems oder infolge einer bedeutenden Änderung in der Datenstruktur notwendig geworden sein.

7.2.4.2 Clientseitige Reaktionen auf die Antwort des Backend-Servers

Mit Hilfe des SSC-Patterns reagiert SLC on AIR bei jedem (erneuten) Zustandekommen einer Online-Konnektivität mit einer Anfrage an das SLC-Backend nach der aktuellen Versionsnummer und erhält die im vorstehenden Abschnitt beschriebene Antwort. Entsprechend der Ausprägung der (empfangenen) Eigenschaft `upgradeReq` reagiert der Client wie folgt:

silent: keine Reaktion

suggested: Der Benutzer wird mit einem Hinweis darauf aufmerksam gemacht, dass eine neue Version vorliegt (siehe Screenshot in Abbildung 7.8). Dieser Hinweis hat jedoch den Charakter einer Empfehlung: auch ein Weiterarbeiten mit der älteren Version bleibt möglich.

mandatory: Die im Screenshot in Abbildung 7.9 gezeigte Reaktion deaktiviert den Client und fordert zum Update auf.



Abbildung 7.8: Screenshot: Hinweis auf die Verfügbarkeit einer neuen Version des RIA-Clients in SLC on AIR

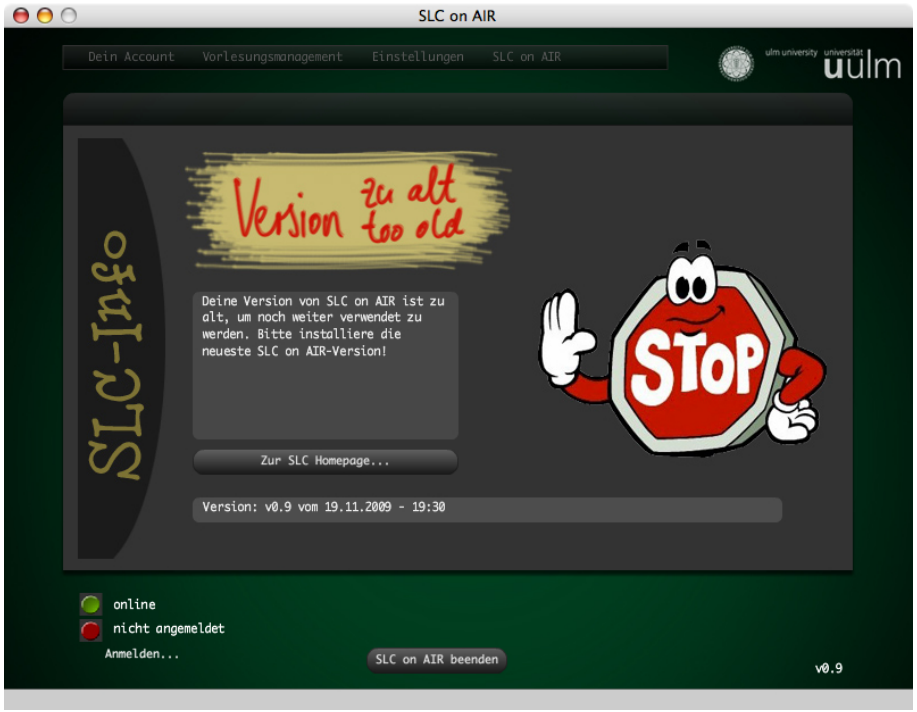


Abbildung 7.9: Screenshot: Die verwendete Version von SLC on AIR ist veraltet und kann nicht weiter verwendet werden.

7.2.4.3 Bezug zum SSC-Pattern

Die zuvor beschriebene Anfrage beim Server erfolgt jedes Mal, wenn sich der Konnektivitäts-Status von *offline* zu *online* ändert. Um dies zu erreichen, wird an dieser Stelle das SSC-Pattern eingesetzt: hier ist es die Anwendung selbst, die *online-sensitiv* gemacht wird und – wie beschrieben – auf detektierte Änderungen im (überwachten) Online-Status reagiert.

Wenn eine neue Version mit `mandatory` gekennzeichnet ist, bedeutet dies *de facto*, dass die lokale (d.h. veraltete) Version von SLC on AIR nicht mehr zu gebrauchen ist, da sie sich jedes Mal, wenn eine bestehende Online-Verbindung erkannt wird, selbstständig deaktiviert. Lediglich ein Offline-Betrieb wäre noch störungsfrei möglich, jedoch widerspricht eine reine Offline-Verwendung dem Charakter einer RIA.

7.2.4.4 Update des RIA-Clients

Wenn sich der Benutzer entscheidet, eine neue Version des RIA-Clients zu installieren, erfolgt die Durchführung der Software-Aktualisierung mit Hilfe des Update-Frameworks von Adobe AIR[14]. Auf diese Weise wird der Benutzer nahezu automatisch durch den gesamten Update-Prozess geführt, ohne dass er selber nennenswerte eigene Bemühungen diesbezüglich unternehmen muss. Wenn eine neue Version auf dem Server gefunden wurde, dann vollzieht sich durch das Update-Framework der gesamte Aktualisierungsprozess (vom Download über die Verifikation der Signatur bis hin zum Installieren des Updates) automatisch. Dies leistet somit auch einen Beitrag zu der mit dem RIA-Client von SLC angestrebten Steigerung an *User Experience*.

7.2.4.5 Ergänzung des AIR-Update-Frameworks

Das AIR-Update-Framework bringt seit Version 1.5 einen eigenen Mechanismus mit, um regelmäßig nach verfügbaren Updates zu suchen. Im Unterschied zu dem hier beschriebenen eigenen Ansatz sieht er jedoch nicht vor, eine Anwendung augenblicklich zu deaktivieren, wenn eine neue Version verfügbar ist. Im Hinblick auf schnelle Reaktionsmöglichkeiten im Fall von Sicherheitsproblemen oder Änderungen in der Datenstruktur erscheint dies im Vergleich zu dem eigenen Ansatz als nachteilig. Ferner lässt sich der AIR-eigene Ansatz nicht in das SSC-Pattern integrieren. Mithin prüft er nicht bei jeder Änderung des Konnektivitäts-Status' auf das Vorliegen einer neuer Version, sondern periodisch nach Ablauf eines vorgegebenen Zeitintervalls. Hier erscheint eine Kombination aus beiden Ansätzen als sinnvoll: einerseits muss bei jedem Aufbau einer Online-Verbindung die Überprüfung erfolgen, andererseits unterstützt auch eine danach periodisch durchgeführte weitere Überprüfung das Bestreben, möglichst unverzüglich auf neue Versionen hinweisen zu können. Der eigene Ansatz ergibt in Kombination mit demjenigen aus dem AIR-Update-Framework somit eine gute gegenseitige Ergänzung.

7.3 Ein Fazit und ein Ausblick zu SLC-β und SLC on AIR

Mit dem RIA-Client von SLC-β (SLC on AIR) liegt ein Praxisbeispiel vor, das die erfolgreiche Anwendung der in Kapitel 6 in einer abstrakten Form erläuterten Ideen (bzw. *Design Pattern*) auf ein reales Software-Projekt und somit die Praxistauglichkeit der beschriebenen

[14] siehe Kapitel 36 in `http://help.adobe.com/en_US/AIR/1.5/devappsflex/devappsflex.pdf`

Ansätze demonstriert. Sein typisches Benutzerklientel setzt sich aus Studierenden zusammen, die – je nachdem, ob sie als Tutoren mitarbeiten oder nicht – über mehr oder weniger viele Berechtigungen verfügen. Dies macht die Anwendung des PAM-Patterns erforderlich.

Internationalen Studierenden soll mit der Unterstützung mehrerer Sprachen entgegengekommen werden, sodass dies die Verwendung des MLL-Patterns in SLC on AIR motiviert hat. Die aktuell vorliegende Version der RIA unterstützt zwar lediglich die deutsche und die englische Sprache, jedoch ermöglich das Konzept des MLL-Patterns eine sehr einfache Erweiterung um jede beliebige weitere Sprache, sodass für das Hinzunehmen weiterer Sprachen keine technischen Hindernisse zu erwarten sind.

Das SSC-Pattern findet im RIA-Client von SLC-β gleich mehrere sinnvolle Einsatz-Szenarien: es dient sowohl zur geeigneten Reaktion auf Änderungen im Konnektivitäts- und Login-Stauts, als auch zur Versionskontrolle und trägt auf diese Weise zur Sicherheit des Systems bei[15].

Immer wieder wurde im Verlauf dieser Arbeit darauf hingewiesen, dass RIAs einen Zugewinn an *Usability* und *User Experience* bringen sollen. Erste Reaktionen einiger Studierender haben gezeigt, dass ihnen die Verwendung des RIA-Clients in der Tat "Spass macht", jedoch sind Aussagen dieser Art stark subjektiver und deshalb nicht verallgemeinerbarer, insbesondere aber auch nicht messbarer Natur. Aus diesem Grund wäre einer der möglichen nächsten Schritte bei der Weiterentwicklung von SLC-β und insbesondere dessen RIA-Clients, eine aussagekräftige Studie über die *Usability* und *User Experience* anzufertigen. Im Rahmen dieser Arbeit, die als primär auf die technische Seite fokussierend ausgelegt ist, wurde eine solche Studie jedoch nicht mehr unternommen.

Die Weiterentwicklung von SLC-β und SLC on AIR adressiert nach aktuellen Planungen nunmehr die Medienintegration, um infolgedessen den Funktionskanon der Systeme um e-Learning-Funktionalitäten erweitern zu können. Erste Implementierungen in diese Richtung wurden bereits erfolgreich unternommen, jedoch tragen sie inhaltlich nichts zu den hier interessierenden Problemstellungen bei, sodass sie hier nicht weiter beschrieben worden sind. Gleichwohl ergeben sich aus der Beschäftigung mit diesen Weiterentwicklungsideen neue Problem- und Fragestellungen, die insoweit also als "*Future Work*" angesehen werden können. Insbesondere die Thematik, ob sich und – wenn ja – wie sich e-Learning-Konzepte und die Technologien der *Augmented Reality* sinnvoll kombinieren lassen, erscheinen in diesem Zusammenhang als hochinteressant.

[15] Dieser Aussage liegt die Annahme zugrunde, dass eventuelle sicherheitsrelevante Fehler im RIA-Client nach ihrer Entdeckung nebst "Fern-Deaktivierung" des Clients mittels der beschriebenen Versionskontrolle keinen Schaden (mehr) verursachen können.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Das *World Wide Web* ist heutzutage ein nahezu unverzichtbarer Bestandteil des öffentlichen Lebens. Immer mehr Informationen und Dienstleistungen werden über "das Web" bezogen, und die rasant fortschreitende Entwicklung der Web-Technologien lässt die Grenzen zwischen einer lokalen (d.h. auf dem eigenen Rechner laufenden) Applikation und einer "im Web" ausgeführten Anwendung mehr und mehr verschwimmen. Web-Anwendungen werden immer interaktiver und multimedialer, sie werden zu *reichhaltigen Internet Anwendungen* (*Rich Internet Applications, RIAs*). Während dies einerseits für viele Menschen einen Zugewinn an Benutzerfreundlichkeit (*User Experience*) bedeutet, gibt es andererseits aber auch jene Gruppe von Internet-Nutzern, die aufgrund von körperlichen und/oder kognitiven Schwächen gar nicht oder nur sehr eingeschränkt von den neuen Entwicklungen im Web profitieren können. Diesen Menschen bleiben die Inhalte entweder ganz verschlossen, oder der Zugang zu den Inhalten weist hohe *Barrieren* auf.

Die Frage, wie Internet-Auftritte, die auf aktuellen Web-Technologien beruhen, ohne derartige Barrieren – mithin also *barrierefrei* – konzipiert und unterhalten werden, steht im Fokus der vorliegenden Arbeit, deren Struktur und wesentlichen wissenschaftlichen Beiträge wie folgt zusammengefasst werden können:

1.) Barrierefreiheit: Darstellung des legislativen und technischen *Status Quo*

Einschlägige Rechtsvorschriften in europäischen und nordamerikanischen Ländern geben der Thematik der Barrierefreiheit eine gewichtige Bedeutung. In Deutschland müssen insbesondere Einrichtungen des öffentlichen Rechts dafür Sorge tragen, dass Ihre Webauftritte barrierefrei sind. Zu Beginn stellt diese Arbeit die entsprechenden Rechtsgrundlagen im In- und Ausland kurz vor.

Sofern Web-Auftritte in "traditioneller" Weise (d.h. im Wesentlichen auf der Grundlage von HTML) unterhalten werden, existieren bereits etablierte und erprobte Lösungen ("*Best Practices*"). Ihrer Bedeutung für die eigenen Lösungen wegen werden auch sie zu Beginn dieser Arbeit erläutert.

2.) Das WebTool – Der Prototyp eines barrierefreien Autorensystems

Obwohl die *Best Practices* mitunter bereits seit mehreren Jahren propagiert werden, scheitert ihre konkrete Verwendung oftmals daran, dass Autoren von Webseiten nicht über die entsprechenden Kenntnisse verfügen. Motiviert von der Idee, barrierefreie Seiten auch unabhängig von individuellen HTML-, CSS- und JavaScript-Kenntnissen erstellen und unterhalten zu können, wurde das *WebTool* als prototypische Implementierung eines Autorensystems vorgestellt, mit dem kleine[1] statische Webauftritte unabhängig von den individuellen Kenntnissen seiner Benutzer oder durch ein ganzes Team von Autoren bzw. Redakteuren erstellt und gepflegt werden können. Das *WebTool* orientiert sich dabei an den Richtlinien des W3C zur Barrierefreiheit von Autorensystemen (*Authoring Tool Accessibility Guidelines*). Durch die Konzentration auf die *Best Practices* und eine kontinuierliche Validierung gegen etablierte Standards trägt das WebTool zur Entstehung barrierefreier Webauftritte bei.

3.) Das jLAF-Framework – Eine Abstraktion des WebTools

Das als kommandozeilenbasiertes Werkzeug konzipierte WebTool lässt sich dergestalt abstrahieren, dass es als *Framework* zur Verfügung steht (“*Java Lightweight Authoring Framework*”, abgekürzt mit jLAF). Mit Hilfe dieses Frameworks kann ein Applikationsserver Inhalte barrierefrei generieren, die zu diesem Zweck aus externen Quellen bezogen werden, ohne dort bereits in einer für den Einsatz im Web aufbereiteten Form vorliegen zu müssen (z.B. in HTML). Dies erleichtert Autoren(teams), die nur geringe oder auch keine Kenntnisse über HTML haben, die Erstellung von barrierefreien und standardkonformen Inhalten. Darüberhinaus erlaubt das jLAF-Framework auch die Unterstützung mehrerer Sprachen.

4.) Statische Rich Internet Applications (RIAs) und barrierefreie HTML-Fallback-Lösungen

Das jLAF-Framework kann zwar wie beschrieben “für sich alleine” eingesetzt werden, jedoch ist seine eigentliche Intention, die automatische Generierung barrierefreier Fallback-Lösungen von statischen RIAs zu ermöglichen. Bei statischen RIAs handelt es sich um eine Unterkategorie der interaktiven und (an *User Experience*) reichhaltigen Web-Anwendungen, bei denen im Wesentlichen die Vermittlung feststehender Inhalte, jedoch nicht so sehr der Applikations-Charakter im Vordergrund steht. Aus den Inhalten statischer RIAs lässt sich mit Hilfe des jLAF-Frameworks automatisiert eine vereinfachte Darstellung erhalten, die im Wesentlichen auf HTML und CSS beruht, sich an entsprechenden Standards orientiert und insoweit als barrierefreie Fallback-Lösung der RIA-Inhalte dient. Durch die mögliche Anreicherung von “unaufdringlichem” (*unobstrusive*) JavaScript und der Berücksichtigung der Paradigmen *Graceful Degradation* bzw. *Progressive Enhancement* kann auch die HTML-Fallback-Lösung um “reichhaltige” Komponenten erweitert werden. Die im jLAF-Framework integrierte Unterstützung von Mehrsprachigkeit wird mit Hilfe des Konzepts des *MultiLanguageLabelers* (MLL) auf RIAs übertragen.

5.) Personalisierung dynamischer Rich Internet Applications

Das Pendant zu statischen RIAs stellen dynamische RIAs dar. Sie lassen sich mit Hilfe des jLAF-Ansatze *nicht* automatisch in eine barrierefreie HTML-Lösung überführen. Andererseits

[1] Eine Definition dieses Attributs findet sich im entsprechenden Teil der Arbeit.

werfen sie ihrerseits Fragen auf, die bei statischen RIAs nicht so sehr im Mittelpunkt stehen. So müssen dynamische RIAs insbesondere in der Lage sein, ihre Präsentation an den momentanen Benutzer anzupassen. Einerseits *kann* dies aufgrund von Vorlieben des Benutzers erfolgen, beispielsweise aufgrund seiner Sprachpräferenzen oder individueller Anforderungen an die optische Aufbereitung. (Dies gilt insoweit analog auch für statische RIAs). Andererseits *muss* dies erfolgen, wenn verschiedene Benutzer über unterschiedliche Berechtigungen verfügen. Denn im Unterschied zu traditionellen Web-Applikationen liegen RIAs stets in einem vollen Funktionsumfang vor, und das auch dann, wenn dieser aufgrund individueller Berechtigungen gar nicht zur Verfügung gestellt werden darf. Dies führt in den Bereich der *Personalisierbarkeit* und *Individualisierbarkeit* von (dynamischen) RIAs. Auch die Änderungen bestimmter "äußerer Umstände" (z.B. der Verlust des Konnektivitäts-Status' oder eine Änderung des Login-Status') machen eine Adaption der RIA an diese neuen Umstände erforderlich. Mit den Konzepten der *Status Sensitive Components* (SSC-Pattern) sowie der *Privilege Aware Menu Structures* (PAM-Pattern) liegen zwei Design-Muster vor, die zur Lösung der beschriebenen Problemstellungen beitragen.

8.2 Ausblick

Die Beschäftigung mit der Thematik der Barrierefreiheit im *World Wide Web* hat zu der Überzeugung geführt, dass diesem Thema in der öffentlichen Wahrnehmung mehr Raum gegeben werden sollte. Durch entsprechende Projekte namhafter Industrieunternehmen und natürlich auch aufgrund der einschlägigen Gesetzgebung sind erste Schritte in diese Richtung bereits unternommen worden.

Mit zunehmender öffentlicher Wahrnehmung wird der Bedarf an Standards wachsen, damit assistive Technologien nicht mehr nur für ein bestimmtes Betriebssystem oder gar nur einen bestimmten Browser entwickelt werden. Eine Untersuchung und Bewertung aktueller Standardisierungs-Vorhaben wäre eine mögliche Fortsetzung dieser Arbeit.

Weitere Ideen für zukünftige Forschungsbeiträge können unter der Thematik "RIAs und *User Experience*" zusammengefasst werden. Was erhöht die *User Experience* bei reichhaltigen Web-Anwendungen tatsächlich? Worin unterscheiden sich in diesem Zusammenhang RIAs von traditionellen (Web-)Applikationen? Wann ist "weniger (Reichhaltigkeit) mehr"? Aus den zunächst sehr subjektiv anmutenden Fragestellungen dieser Art lassen sich Studien mit Benutzern entwickeln, aus deren Ergebnissen Hilfestellungen für eine (zielgruppengerechte) Darstellung von Internet-Auftritten gewonnen werden können. Sieht man den Einsatz von RIAs in einem Marketing-Kontext (beispielsweise, um interaktive Prospekte bereitzustellen), sind aus derartigen Untersuchungen zudem aus ökonomischen Gesichtspunkten hochinteressante Ergebnisse zu erwarten.

Wie am Ende des Kapitels 7 angedeutet, ist ein Ausbau von SLC-βim Hinblick auf dessen e-Learning-Kapazitäten angedacht. Hieraus ergeben sich Fragestellungen, die einerseits technischer Natur sind (und beispielsweise das Zusammenspiel von RIA-Client und Backend-Server adressieren), darüberhinaus erneut aber auch die Unterstützung von Barrierefreiheit tangieren. Neben derartigen Fragestellungen erscheint der Gedanke, den RIA-Client "SLC on AIR" mit Hilfe von Ansätzen der *Augmented Reality* zu einer besonders reichhaltigen e-Learning-Lösung zu erweitern, sehr reizvoll. Eine Evaluation über die Chancen, die eine solche Erweiterung bietet, ist als unmittelbare Fortsetzung dieser Arbeit in Planung.

Summary

Nowadays, the World Wide Web (WWW) is an essential part of everyone's daily life. The amount of information and services that one obtains from "the web" is growing each day, and the emerging web technologies made today's web applications become indistinctable from traditional desktop applications. Web applications nowadays are *richer* than they ever were before—they are *Rich Internet Appications* (RIAs).

However, rich internet applications may remain inaccessible for people with either physical or cognitive disabilities. Due to legislation, in some countries (e.g. the USA and within the EU) public institutions are forced to have their websites built in a way that they are *accessible* for everyone—no matter whether he or she is disabled or not.

The present thesis covers topics on Accessibility and Rich Internet Applications. After an introduction in **Chapter 1**, the status quo of the legislation regarding (website) accessibility in the USA as well as in the EU is presented in **Chapter 2**. Thereafter, an overview of *Best Practices* in traditional web design (this is, using HTML, CSS and JavaScript) is given.

In **Chapter 3**, a prototype for a command line based authoring system is introduced which can be used by authors (and teams of authors) to build and maintain small *accessible* websites, even if they neither have a good knowledge of HTML nor other related skills. The authoring system is named *WebTool*. Further in Chapter 3, the command line based *WebTool* is transferred to a framework (called jLAF) that is intended to run on an application server. The jLAF framework can be used to generate web pages *on the fly*, but it also serves as a basis for later work.

Chapter 4 presents an overview of current RIA technologies, in particular with respect to their support of accessibility. This will show that a sufficient solution for obtaining accessible Rich Internet Applications does not exist, yet.

The jLAF framework will be expanded in **Chapter 5** in order to provide a mechanism for retrieving an HTML version automatically from a *static* Rich Internet Applications. Static RIAs represent a specific subset of RIAs with only an *informatory* character. Those RIAs are not real applications but rather multimedia-based rich websites. This means that the content that is displayed is not driven by the users' input but rather is received from pre-existing files or a database. (For example, catalogs and brochures may be presented as a static rich internet application). Because of their pre-existing content, those RIAs can be transformed easily to accessible, web-standards based HTML fallback versions. Furthermore, it is shown how this type of content can be provided in multiple language. To prove the suitability of the presented solutions, Chapter 5 closes with the presentation of the *Webbook* upon which each of the ideas has been applied successfully.

RIAs that are *dynamic* rather than static (that is, real (rich) internet *applications*) can not be transformed to an HTML fallback version by means of the solution in Chapter 5. Instead, those RIAs can be customized to each user's requirements (e.g. the language setting, or the layout

preferences). However, sometimes a dynamic RIA *must be* customized to the user. This is because a RIA is always "one piece of software", containing each and every functionality, even if the current user's privileges are insufficient to use each of these functionalities. As a solution for that type of problem, the concept of *Privilege Aware Menu Structures* (PAM pattern) is provided in **Chapter 6**. Related to that concept is the idea of *Status Sensitive Components* (SSC pattern), which is also presented in Chapter 6.

The successful application of the PAM pattern and the SSC pattern as well as a way to support multiple languages easily are demonstrated in a practical example in **Chapter 7**. Using the example of the management software for students and lectures at the faculty of mathematics and economics at Ulm University, the utilization of the presented patterns onto real world scenarios is proven.

The thesis is concluded with a summary and an overview over future work in **Chapter 8**.

Anhang: Übersicht über JavaScript-Frameworks

In Abschnitt 4.2.1 auf Seite 84 wird eine Webseite referenziert, auf der ein Vergleich von zum Zeitpunkt der Entstehung dieser Arbeit aktuellen JavaScript-Frameworks nachzulesen ist. Dieser Vergleich zählt auf, welche JavaScript-Frameworks welche reichhaltigen GUI-Elemente unterstützen.

Die nachfolgende Aufzählung fasst die im Vergleich genannten Frameworks zusammen. Weil der Vergleich bereits im Januar 2009 erschienen ist, wurde die Aufzählung zum Zwecke der Aktualisierung um ein weiteres Framework ergänzt. Sie ist alphabetisch sortiert, d.h. die Reihenfolge gibt nicht ihre Bedeutung wieder. Sofern nicht anders angegeben, stehen die Frameworks kostenlos zur Verfügung (Stand: November 2009).

Tabelle 8.1: Aufzählung aktueller JavaScript-Frameworks (Nov. 2009)

Name des Frameworks	URL	Bemerkungen
Backbase	`http://www.backbase.com/products/enterprise-ajax/`	kommerzielles Produkt
Dojo	`http://www.dojotoolkit.org/`	
Echo2	`http://echo.nextapp.com/site/echo2`	
ExtJS	`http://www.extjs.com/`	
Google Web Toolkit	`http://code.google.com/intl/de/webtoolkit/`	
IT Mill	`http://www.itmill.com/`	Server-zentrierter Ansatz
JQuery	`http://jquery.coma`	in Verbindung mit Bibliothek namens Interface zu sehen
LivePipeUI	`http://livepipe.net/`	Sammlung von *Widgets* für das Prototype Framework
MochaUI	`http://mochaui.com/demo/`	
MooTools	`http://mootools.net/`	
Prototype	`http://www.prototypejs.org/`	
script.aculo.us	`http://script.aculo.us/`	basiert auf dem Prototype Framework
SproutCore	`http://www.sproutcore.com/`	
Spry	`http://labs.adobe.com/technologies/spry/`	
YUI	`http://developer.yahoo.com/yui/`	YUI = Yahoo! UI Library

Diese Liste der Frameworks erhebt nicht den Anspruch, vollständig zu sein. Ebenso kann nicht von allen erwähnten Frameworks erwartet werden, sich am Markt dauerhaft zu behaupten. Insoweit ist die Liste als Status quo zum Ende des Jahres 2009 zu sehen.

Publikationsliste

- **Norbert Heidenbluth und Franz Schweiggert**
  Privilege Aware Menus: How to adapt menu structures of Rich Internet Applications to user privileges.
  In: Proceedings of the International Conference on Enterprise Information Systems and Web Technologies (EISWT 2009), ISRST, 2009, S. 28-35

- **Norbert Heidenbluth und Franz Schweiggert**
  Status Sensitive Components: Adapting Rich Internet Applications to their Runtime Context.
  In: ICDS, IEEE Computer Society, 2009, S. 133-138

- **Boto Bako, Andreas Borchert, Norbert Heidenbluth und Johannes Mayer**
  Linearly Ordered Plugins through Self-Organization.
  In: ICAS, IEEE Computer Society, 2006. – ISBN 0–7695–2653–5, S. 8-15

- **Boto Bako, Andreas Borchert, Norbert Heidenbluth und Johannes Mayer**
  Plugin-Based Systems with Self-Organized Hierarchical Presentation.
  In: Arabnia, Hamid R. (Hrsg.) ; Reza, Hassan (Hrsg.): Software Engineering Research and Practice, CSREA Press, 2006. – ISBN 1–932415–91–2, S. 577-584

Danksagungen

Das Anfertigen dieser Arbeit wäre nicht möglich gewesen, wenn mich nicht viele Menschen dabei unterstützt hätten. Ihnen allen gebührt mein aufrichtiger Dank.

So möchte ich mich an erster Stelle ganz herzlich bei Prof. Dr. Franz Schweiggert bedanken. An seinem Institut für Angewandte Informationsverarbeitung entstand diese Arbeit. Prof. Schweiggert hat während meines Studiums meine Begeisterung für die Informatik geweckt, er gab mir die Chance, an seinem Institut mitzuarbeiten, dort meine Promotion auf den Weg zu bringen, und er stand mir als wertvoller Betreuer in allen Belangen jederzeit zur Seite. Seine großzügige Unterstützung und die freundschaftliche Arbeitsatmosphäre an seinem Institut haben den Fortschritt dieser Arbeit maßgeblich beeinflusst.

Mein ebenso herzlicher Dank geht an PD Dr. habil. Johannes Mayer. Als hilfsbereiter Kollege und geduldiger Mentor hat er mich bei meinen ersten Schritten in die Forschung begleitet, und er war mir selbst dann noch ein Ratgeber und Ansprechpartner, als ihn sein Weg bereits aus Ulm fort geführt hatte. Ich bedanke mich bei ihm insbesondere dafür, dass er sich sofort bereiterklärt hat, als Gutachter für diese Arbeit zur Verfügung zu stehen.

Ebenfalls sofort bereit, diese Arbeit zu begutachten, war Prof. Dr. Wolfgang Minker vom Institut für Dialogsysteme an der Universität Ulm. Auch ihm möchte ich an dieser Stelle dafür herzlich danken.

Für die Zusammenarbeit an zwei Veröffentlichungen geht mein Dank auch an Herrn Dr. Andreas Borchert und Herrn Dipl.-Inf. Boto Bako. Ebenso möchte ich mich bei Frau Ljubow Rutzen-Loesevitz für ihre Hilfe beim Übersetzen meiner Veröffentlichungen bedanken.

Ein weiterer Dank geht an alle ehemaligen und aktuellen Kolleginnen und Kollegen am Institut für Angewandte Informationsverarbeitung. Ob in konstruktiver Zusammenarbeit oder bei gemeinsamen Freizeitaktivitäten: in ihnen habe ich stets hilfsbereite Freunde und ein echtes "Team" gefunden, und von ihnen habe ich nicht nur fachlich, sondern auch für das Leben gelernt.

Neben meinen Kolleginnen und Kollegen haben mich noch viele weitere Menschen während der vergangenen Jahre mit Rat und Tat begleitet. Ihnen allen gebührt mein Dank, auch wenn ich sie nicht einzeln aufzähle, um nicht Gefahr zu laufen, den einen oder die andere dabei zu vergessen.

Von ganzem Herzen geht mein Dank an meine Eltern. Sie haben mich zu jeder Zeit in allen meinen Vorhaben liebevoll unterstützt, und ohne sie wären mein Studium, das Anfertigen dieser Arbeit sowie darüberhinaus noch vieles mehr nicht möglich gewesen. Über Erfolge haben wir uns stets gemeinsam gefreut, und in schwierigen Phasen konnte und kann ich auf ihre Begleitung bauen. Ihnen möchte ich diese Arbeit widmen.

Literaturverzeichnis

[Ado08] Adobe Systems Incorporated: *BlazeDS Developer Guide*. Version: 2008. http://livedocs.adobe.com/blazeds/1/blazeds_devguide/, Abruf: 15.11.2009

[Ado09a] Adobe Systems Incorporated: *Accessibility best practices for Flex*. Version: 2009. http://www.adobe.com/accessibility/products/flex/best_practices.html, Abruf: 2009-11-14

[Ado09b] Adobe Systems Incorporated: *Flex accessibility overview*. Version: 2009. http://www.adobe.com/accessibility/products/flex/overview.html, Abruf: 03.11.2009

[BBHM06a] Bako, Boto ; Borchert, Andreas ; Heidenbluth, Norbert ; Mayer, Johannes: Linearly Ordered Plugins through Self-Organization. In: *ICAS*, IEEE Computer Society, 2006. – ISBN 0–7695–2653–5, S. 8

[BBHM06b] Bako, Boto ; Borchert, Andreas ; Heidenbluth, Norbert ; Mayer, Johannes: Plugin-Based Systems with Self-Organized Hierarchical Presentation. In: Arabnia, Hamid R. (Hrsg.) ; Reza, Hassan (Hrsg.): *Software Engineering Research and Practice*, CSREA Press, 2006. – ISBN 1–932415–91–2, S. 577–584

[Ber04] Bergsten, Hans: *JavaServerPages*. O'Reilly, 2004

[Bla09] Blaxill, Alistair: What's the difference between personalisation and individualisation? In: *Technology Weekly* (2009). – June 2006

[Bol04] Bolin, Michael: *TargetAlert*. Version: 2004. http://www.bolinfest.com/targetalert/download.html, Abruf: 24.10.2009

[CF03] Champeon, Steven ; Finck, Nick: *Inclusive Web Design For The Future With Progressive Enhancement*. Version: 03 2003. http://www.hesketh.com/publications/inclusive_web_design_for_the_future, Abruf: 24.11.2009. SXS Week 2003, Austin, TX, USA

[Dav04a] Davies, Mike: *Accessibility and Content Management*. Version: 04 2004. http://www.isolani.co.uk/blog/access/AccessibilityAndContentManagement, Abruf: 25.10.2009

[Dav04b] Davies, Mike: *Accessibility of text-only websites*. Version: 2004. http://www.isolani.co.uk/articles/accessibilityOfTextOnlyWebsites.html, Abruf: 15.11.2009

[DH09] Dashorst, Martijn ; Hillenius, Eelco: *Wicket in Action*. Manning, 2009

[ECM99] ECMA International: *ECMAScript Language Specification*. Version: 1999. http://www.ecma-international.org/publications/standards/ecma-262.htm, Abruf: 15.11.2009

[EGK06] Ebersbach, Anja ; Glaser, Markus ; Kubani, Radovan: *Joomla! – Das Handbuch für Einsteiger*. Galileo Computing, 2006

[Gar02] Garrett, Jesse J.: *The Elements of User Experience: User-Centered Design for the Web*. New Riders Press, 2002

[Gar05] Garrett, Jesse J.: *Ajax: A New Approach to Web Applications*. Version: 02 2005. http://www.adaptivepath.com/ideas/essays/archives/000385.php, Abruf: 24.10.2009

[GHJV94] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994

[Gib06] Gibson, Becky: *AJAX Accessibility Overview*. Version: 04 2006. http://www-03.ibm.com/able/resources/ajaxaccessibility.html, Abruf: 24.10.2009

[Hei04a] Heidenbluth, Norbert: *Zur Personalisierung von Navigationsstrukturen in Portalen*, Universität Ulm, Diplomarbeit, 2004

[Hei04b] Hein, Ansgar: *Content Management Systeme im Einsatz für Barrierefreies Internet*. Version: 04 2004. http://www.contentmanager.de/magazin/artikel_481_cms_fuer_barrierefreiheit.html, Abruf: 25.10.2009

[Hei05] Heilmann, Christian: *Unobstrusive JavaScript*. Version: 2005. http://www.onlinetools.org/articles/unobtrusivejavascript/, Abruf: 15.11.2009

[HS09a] Heidenbluth, Norbert ; Schweiggert, Franz: Privilege Aware Menus: How to adapt menu structures of Rich Internet Applications to user privileges. In: *Proceedings of the International Conference on Enterprise Information Systems and Web Technologies (EISWT 2009)*, ISRST, 2009, S. 28–35

[HS09b] Heidenbluth, Norbert ; Schweiggert, Franz: Status Sensitive Components: Adapting Rich Internet Applications to Their Runtime Context. In: *ICDS*, IEEE Computer Society, 2009, S. 133–138

[HW04] Hohpe, Gregor ; Woolf, Bobby: *Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions*. Addison Wesley, 2004

[Kei] Keith, Jeremy: *Progressive Enhancement with Ajax*. http://adactio.com/journal/959, Abruf: 18.11.2009. – Adactio: Journal

[Kor02] Korpela, Jukka: *Augmentative Authoring – A different look at "graceful degradation" in Web authoring*. Version: 08 2002. http://www.cs.tut.fi/~jkorpela/html/augm.html, Abruf: 15.11.2009

[Kra05] Krantz, Peter: *AJAX and Accessibility*. Version: 03 2005. http://www.standards-schmandards.com/2005/ajax-and-accessibility/, Abruf: 24.10.2009

[Kru05] Krug, Steven: *Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition.* New Riders Press, 2005

[Lan08] Lange, Thomas M.: *Standardkonforme Integration von Rich Internet Applications in aktuelle Portaltechnologien*, Universität Ulm, Diplomarbeit, 2008

[Lem08] Lemon, Gez: *Introduction to WAI ARIA.* Version: 08 2008. `http://dev.opera.com/articles/view/introduction-to-wai-aria/`, Abruf: 10.11.2009

[Lin04] Linderman, Matthew: *Web Interface Design Tip: The Yellow Fade Technique.* Version: 02 2004. `http://www.37signals.com/svn/archives/000558.php`, Abruf: 25.10.2009

[Luč08] Lučić, Zdravko: *Entwicklung plattformunabhängiger Rich Internet Applications mit OpenLaszlo*, Universität Ulm, Diplomarbeit, 2008

[MCG09] Martins, Jos ; Cruz, Jos ; Gonalves, Ramiro: Portuguese Web Accessibility – Portuguese Enterprises Websites Accessibility Evaluation. In: *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS 2009)* (2009), S. 142–147

[MDBM02] Manko, Jennifer ; Dey, Anind ; Batra, Udit ; Moore, Melody: Web Accessibility for Low Bandwidth Input. In: *Proceedings of the fifth international ACM conference on Assistive technologies* (2002), S. 17–24

[Mel02] Melzer, Ingo: *An Abstraction to Implement Internet Portals*, Universität Ulm, Diss., 2002

[MFP06] Mühl, Gero ; Fiege, Ludger ; Petzuch, Peter: *Distributed Event-based Systems.* Springer, 2006

[Mü07] Münz, Stefan: *SelfHTML.* Version: 03 2007. `http://de.selfhtml.org`, Abruf: 24.10.2009

[Mü09] Müller, Florian: *Professionelle Rich-Client-Lösungen mit Flex und Java.* Addison-Wesley, 2009

[Ols07] Olsson, Tommy: *Graceful Degradation & Progressive Enhancement.* Version: 02 2007. `http://accessites.org/site/2007/02/graceful-degradation-progressive-enhancement/`, Abruf: 15.11.2009

[O'R05] O'Reilly, Tim: *What is Web 2.0.* Version: 10 2005. `http://www.oreilly.de/artikel/web20.html`. – 1 S.

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

[Reg05] Regan, Bob: *Best Practices for Accessible Flash Design.* Version: 2005. `http://www.adobe.com/resources/accessibility/best_practices/best_practices_acc_flash.pdf`, Abruf: 24.10.2009

[Rog07] Rogowski, Ron: *The Business Case For Rich Internet Applications.* Forrester Research Report, 12 2007

[Rog08] Rogowski, Ron: *Integrate RIAs Into User Experiences.* Forrester Research Report, 06 2008

[Sch08] Schonfeldt, Erick: *TechCrunch Weblog.* Version: 06 2008. http://www.techcrunch.com/2008/06/30/once-nearly-invisible-to-search-engines-flash-files-can-now-be-found-and-indexed/, Abruf: 03.11.2009

[Smi07] Smith, Gordon: *Flex 3:Feature Introductions: Runtime Localization.* Version: 09 2007. http://labs.adobe.com/wiki/index.php/Flex_3:Feature_Introductions:_Runtime_Localization, Abruf: 15.11.2009

[SN09] Scott, Bill ; Neil, Theresa: *Designing Web Interfaces: Principles and Patterns for Rich Interactions.* O'Reilly, 2009

[Spr09] Spring Source: *Spring BlazeDS Integration Home.* Version: 2009. http://www.springsource.org/spring-flex, Abruf: 19.11.2009

[ST09] Shinohara, Kristen ; Tenenberg, Josh: A Blind Person's Interactions with Technology. In: *Communications of the ACM* 08/09 Vol. 52, No. 08 (2009), S. 58–66. http://dx.doi.org/10.1145/1536616.1536636. – DOI 10.1145/1536616.1536636

[SUN] SUN Microsystems: *JavaScript Technology.* http://java.sun.com/javascript/, Abruf: 24.10.2009

[Thu07] Thurow, Shari: *Search Engine Visibility, Second Edition.* Peachpit Press, 2007

[TLM09] Teles de Lima, Sinésio ; Lima, Fernanda ; Marçal de Oliveira, Káthia: Evaluating the Accessibility of Websites to Define Indicators in Service Level Agreements. In: *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS 2009)* (2009), S. 858–869

[TNS09] TNS EMNID: *Umfrage: Bekanntheit des Begriffs "Barrierefreie Websites".* Version: 05 2009. http://www.stiftung-barrierefrei-kommunizieren.de/front_content.php?idcat=29, Abruf: 03.11.2009

[U.S98] U.S. Department of Justice: *Workforce Investment Act.* Version: 1998. http://www.usdoj.gov/crt/508/508law.php, Abruf: 24.10.2009

[Utaa] Utah State University, Center for Persons with Disabilities: *Creating Accessible Macromedia Flash Content.* http://www.webaim.org/techniques/flash/, Abruf: 24.10.2009

[Utab] Utah State University, Center for Persons with Disabilities: *WebAIM.* http://webaim.org, Abruf: 24.10.2009

[Utac] Utah State University, Center for Persons with Disabilities: *WebAIM.* http://www.webaim.org/intro/#principles, Abruf: 24.10.2009

[W3C99] W3C: *HTML 4.01 Specification.* Version: 12 1999. http://www.w3.org/TR/REC-html40/

[W3C08a] W3C: *Authoring Tool Accessibility Guidelines 2.0.* Version: 12 2008. http://www.w3.org/TR/2008/REC-WCAG20-20081211/, Abruf: 25.10.2009

[W3C08b] W3C: *Understanding WCAG 2.0 - A guide to understanding and implementing WCAG 2.0.* Version: 12 2008. http://www.w3.org/TR/2008/REC-WCAG20-20081211/, Abruf: 24.10.2009

[W3C09] W3C: *HTML 5—A vocabulary and associated APIs for HTML and XHTML.* Version: 10 2009. `http://www.w3.org/TR/html5/`, Abruf: 03.11.2009. – W3C Working Draft

[Wik] WIKIPEDIA: *Möglichkeiten zur Formatierung von Wikipedia-Artikeln (aus der Wikipedia Online Hilfe).* `http://de.wikipedia.org/wiki/Hilfe:Textgestaltung`, Abruf: 25.10.2009

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig angefertigt habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe.

Ulm, den 18. Januar 2010

_____
(Norbert Heidenbluth)