Hochschule für
Technik und Wirtschaft
Dresden
University of Applied Sciences

Fachbereich Informatik/Mathematik

DIPLOMARBEIT

im Studiengang Medieninformatik

Thema:

Gegenüberstellung der Technologien HTML5/JavaScript und des Flex-Frameworks anhand der prototypischen Umsetzung einer Medienverwaltung

eingereicht von	Mathias Brodala
Matrikelnummer	22635
eingereicht am	30.06.2011
Betreuer	Prof. Dr. Teresa Merino

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

AIR
: Adobe Integrated Runtime

AMF
: Action Message Format

API
: Application Programming Interface

AS3
: Adobe ActionScript 3.0

BLOB
: Binary Large Object

CERN
: Europäische Organisation für Kernforschung

Codec
: Coder-Decoder

CORS
: Cross Origin Resource Sharing

CSS
: Cascading Style Sheets

DAP
: Device APIs and Policy Working Group

DEFLATE
: Algorithmus zur verlustfreien Datenkompression

DOM
: Document Object Model

DSP
: Digital Signal Processor

EBNF
: Erweiterte Backus-Naur Form

ECMA
: European Computer Manufacturers Association

FFT
: Fast Fourier Transformation

FLV
: F4V
: Flash Video

GTK
: GIMP Toolkit

hixie
: Kürzel von Ian Hickson

HTML
: Hypertext Markup Language

HTTP
: Hypertext Transfer Protocol

ID

Identifier

IDE

Integrated Development Environment

IDL

Interface Definition Language

IETF

Internet Engineering Task Force

IP

Internet Protocol

JS

JavaScript

JSON

JavaScript Object Notation

LSO

Local Shared Objects

MDC

Mozilla Developer Center

MIME

Multipurpose Internet Mail Extensions

MXML

Macromedia eXtensible Markup Language

OpenGL

Open Graphics Library

OSMF

Open Source Media Framework

PHP

Hypertext Preprocessor

RFC

Request for Comments

RIA

Rich Internet Applications

SDK

Software Development Kit

SHA-1

Secure Hash Algorithm 1

SQL

Structured Query Language

SSL

Secure Socket Layer

SWF

Shockwave Flash, Small Web Format

TCP

Transport Control Protocol

UA

User Agent

UI

User Interface

UIA
: User Initiated Action

URI
: Uniform Resource Identifier

URL
: Uniform Resource Location

VML
: Vector Markup Language

W3C
: World Wide Web Consortium

WebGL
: OpenGL ES 2.0 for the Web

WebM
: Offener Standard zur Verbreitung von Mediendateien

WHATWG
: Web Hypertext Application Technology Working Group

WS
: WebSocket

XML
: eXtensible Markup Language

0 Einleitung

In den letzten Jahren haben Browserhersteller im ständigen Wettbewerb untereinander und immer im Bestreben, ihren Nutzern neue Möglichkeiten in Bezug auf reiche Inhalte im Web und neue Wege der Interaktion zu bieten, vermehrt neue Technologien entwickelt. Nicht immer ist dieser Prozess offen vonstatten gegangen und mündete in verschiedene proprietäre Umsetzungen. Zudem wurden häufig verschiedene, zueinander inkompatible, Ansätze für die gleichen Probleme implementiert. Gerade bei einem geschlossenen Entwicklungsprozess beschränkt sich die Verbreitung der betreffenden Implementierung damit zwangsläufig auf einen bestimmten Browser.

Zugleich hat sich jedoch gezeigt, dass es sinnvoll ist, geschlossen entwickelte Ideen zu veröffentlichen und offen zu dokumentieren. Hierdurch entsteht die Möglichkeit alternativer Implementierungen, wodurch ein ständiger Austausch und eine Optimierung auf das Ziel ermöglicht wird. Das Ziel ist in den allermeisten Fällen schlussendlich das Web für Nutzer attraktiv zu machen.

Während das World Wide Web zu Anfangszeiten noch fast ausschließlich von technisch versierten Nutzern bevölkert wurde, hat sich diese Entwicklung in den letzten Jahren stark gewandelt. Heute entdecken immer mehr einfache Nutzer das Web als Informationsquelle, als Schauplatz der eigenen Darstellung und Kommunikationsmedium mit anderen Nutzern für sich. In diesem Sinne hat sich auch der Umgang in Bezug auf die Benutzerinteraktion gewandelt. Der Fokus liegt heute ganz klar auf einfachen, intuitiven Oberflächen, welche den Nutzer ohne Ablenkung schnell an das gewünschte Ziel führen. Daher ist es unverzichtbar, dass viele frühere Beschränkungen im Kontext Web entfallen und durch gut durchdachte Möglichkeiten ersetzt werden.

Die in dieser Arbeit betrachteten Technologien HTML5 und JavaScript entsprechen diesem Vorhaben und standardisieren langjährig unabhängig entwickelte Lösungen nicht jedoch ohne sie gleichzeitig um weitere wichtige Funktionen zu erweitern. Das Flash-Plugin ist bislang eine übliche Lösung, um erweiterte Funktionalität im Web zu ermöglichen. Aus diesem Grund bietet sich das Flex-Framework als von Adobe eigenständig entwickelte Lösung zum Vergleich an.

0.1 Zielstellung

Im Rahmen dieser Diplomarbeit wird mit Hilfe der beiden Technologien HTML5/Javascript und dem Adobe Flex-Framework der Prototyp einer Medienverwaltung implementiert. Die beiden genannten Technologien werden hierbei anhand einiger für diesen Prototyp relevanten Gesichtspunkte gegenüber gestellt. Das Ziel soll hier nicht nur sein, einzelne Techniken innerhalb dieser Technologien zu beleuchten, sondern dies in einer

zusammenhängenden Applikation zu vollziehen.

Dabei soll eine grafische Benutzeroberfläche entwickelt werden, welche in sich konsistent Zugriff auf die verschiedenen Aspekte ermöglicht, ohne sie spezifisch in den Vordergrund zu stellen. Das Hauptaugenmerk liegt hier auf der transparenten und nahtlosen Integration der verschiedenen Techniken. Bewährte Bedienelemente z.B. zur Bewertung von Ressourcen sollen unter Verwendung der neuen Techniken implementiert werden.

Die in dem Prototypen zu verwaltenden Medien bzw. Ressourcen sind hierbei nicht nur von lokalen sondern auch von fremden, domainübergreifenden Quellen zu beziehen. Da eine immer stärkere Vernetzung von Nutzern untereinander zu beobachten ist, ist dieser Aspekt unverzichtbar. Inwieweit bisher stets vorgebrachte Sicherheitsbedenken beseitigt oder diesen entsprochen werden kann, wird in der vorliegenden Arbeit beleuchtet.

Zum Übertragen von Ressourcen zu und von Quellen soll ein verstärktes Augenmerk auf eine Interaktion mittels Drag-and-Drop gelegt werden. Während diese Form der Interaktion in den letzten Jahren im Kontext Web je nach Technologie noch eher eingeschränkt möglich war, soll nun durch die Möglichkeiten der neuen Spezifikationen und Implementierungen beleuchtet werden, inwieweit deren Einsatz im alltäglichen Web verbessert werden kann. Dies ist nicht auf Drag-and-Drop-Operationen zwischen Quellen beschränkt, sondern dehnt sich auch auf die Übertragung von Dateien vom lokalen Speicher zu Quellen aus. Während das Hochladen von Dateien bisher bereits dialogbasiert möglich war, soll der Prozess in diesem Zuge deutlich vereinfacht werden.

Als Alternative zur Kommunikation über das zustandslose HTTP werden auch Socket-basierte Verbindungen implementiert, um persistente Verbindungen zwischen zwei Punkten zu schaffen, welche eine bidirektionale Kommunikation erlauben. Gerade in diesem Punkt ist bis jetzt im Kontext Web nur eine simulierte Kommunikation über Umwege möglich, wie ein regelmäßiges Abfragen von Zuständen (Polling) oder dem Aufrechterhalten von Verbindungen bzw. stetigem Öffnen einer neuen Verbindung zur Verarbeitung potentieller Nachrichten (Comet) [Russel06].

Schließlich sollen in Hinsicht auf die eher beschränkte Wiedergabe von Medientypen in HTML-Dokumenten neue Möglichkeiten erkundet werden, konkret um Audio- und Videowiedergabe zu ermöglichen. Bisher beschränkt sich dies auf eingebettete Alternativlösungen, wozu das Flash-Plugin von Adobe und darin bspw. Adobe Flex zählen. Inwieweit dies nunmehr nativ und ohne Zusatzprogramme möglich ist, soll aufgeführt und implementiert werden.

Zu guter Letzt erscheint interessant, ob durch neue Techniken die Art und Menge der lokal persistierbaren Informationen erweitert werden kann. Sowohl HTML als auch Adobe Flex über das Adobe Flash-Plugin verfügen bereits jetzt über mehr oder weniger beschränkte Möglichkeiten, Informationen lokal und über mehrere Sitzungen hinweg zu speichern. Eine

Verbesserung in dieser Hinsicht erlaubt eine verstärkte Anpassung an die eigenen Vorlieben und spart Kommunikationswege.

0.2 Aufbau

Im Folgenden werden für diese Diplomarbeit relevante Begriffe erläutert, die dem Verständnis der Sachverhalte dienlich sind. In diesem Zuge wird auch kurz auf die historische Entwicklung der beiden Technologien HTML5/Javascript und Adobe Flex sowie das für beide relevante CSS eingegangen.

Es folgt ein Überblick über die Anforderungen und den strukturellen Aufbau des Prototypen, welcher in den Implementierungen der beiden Technologien als Leitfaden dienen soll. Die Vergleichbarkeit der verschiedenen Implementierungen soll hierdurch vereinfacht werden, womit eine Beschränkung auf die tatsächlichen Eigenheiten möglich ist. Ein allzu unterschiedlicher Programmaufbau und -ablauf wäre diesem Vorhaben nicht dienlich. Hieran schließen fünf Kapitel an, welche sich auf die einzelnen eingangs genannten funktionellen Schwerpunkte konzentrieren und dahingehend die konkreten Implementierungen innerhalb der beiden Technologien betrachten, erläutern und bewertend gegenüber stellen. Es wird auf die Eigenheiten bei der Konzeptionierung sowie Vorzüge und Nachteile bei der eigentlichen Umsetzung eingegangen sowie eine Einschätzung ergänzt, in welcher Hinsicht hier Potential zu Verbesserungen besteht.

Schließlich werden die in den beiden implementierten Prototypen eingesetzten Techniken zusammenfassend beleuchtet und ein Fazit in Bezug auf den Aufwand und die nutzbaren Möglichkeiten gebildet. Es folgt ein Ausblick auf die weitere Entwicklung in naher und ferner Zukunft sowie wünschenswerte Aspekte.

1 Begriffserklärungen

In diesem Kapitel werden grundsätzliche Begriffe erklärt, die dem Verständnis in Bezug auf die in dieser Diplomarbeit beleuchteten Technologien dienlich sind. Auf den Grundgedanken der betreffenden Technologie wird ebenso eingegangen wie auf die bisherige Entwicklung und Ausrichtung.

1.1 HTML5

Abbildung 1.1: HTML5-Logo

Das Standardformat für die textliche Strukturierung und semantische Darstellung von Inhalten im World Wide Web präsentiert sich mit HTML5 in seiner neuesten Version. Entwickelt und erstmalig vorstellt von Tim Berners-Lee am 13. März 1989 am CERN in Genf durchlief das HTML-Format eine umfangreiche Entwicklung und wurde stets an die steigenden Anforderungen bezüglich reicher Inhalte im Web angepasst und weiterentwickelt. Diesbezüglich gab es beim Entwicklungsprozess von HTML5 jedoch Uneinigkeiten. Während für die Entwicklung aller HTML-Version bis einschließlich Version 4 und die Neuformulierung in XML in Form von XHTML 1.0 und XHTML 1.1 noch das World Wide Web Consortium [W3C] die treibende Kraft war, breitete sich bei einigen Browserherstellern und Entwicklern Unmut ob des unflexiblen und langsamen Entwicklungsmodells aus. Die Web Hypertext Application Technology Working Group [WHATWG] wurde am 04. Juni 2004 gegründet und befasste sich mit einem offeneren und schnelleren Entwurf einer neuen HTML-Version. Diese sollte sich verstärkt an den tatsächlichen Bedürfnissen von Nutzern und Entwicklern im Web orientieren und schließlich beim W3C als Standard eingereicht werden. Die Bemühungen mündeten in der Spezifikation Web Applications 1.0 [WebApps]. Während Tim Berners-Lee zu Beginn noch eine parallele Entwicklung von Web Applications 1.0, nun umbenannt in HTML5, und dem hauseigenen Format XHTML 2.0 ankündigte[1], brach das W3C später sein Bestreben ab[2], XHTML 2.0 als Standard im Web durchzusetzen und verschob den Fokus auf die Vollendung der HTML5-Spezifikation.

Unabhängig von der Arbeit des W3C versteht es die WHATWG nach wie vor als ihre Aufgabe, den HTML-Standard stetig weiterzuentwickeln, was sich kürzlich in der Entfernung der Versionsnummer „5“ aus dem Namen verdeutlichte [HTML-WHATWG]. Durch diesen Schritt soll deutlich gemacht werden, dass es sich bei HTML nicht um eine Spezifikation handelt, die

[1] http://dig.csail.mit.edu/breadcrumbs/node/166 (abgerufen am 02.06.2011)

[2] http://www.w3.org/News/2009#item119 (abgerufen am 02.06.2011)

in unterschiedlichen Abständen nach Änderungen in einer neuen Version veröffentlicht und damit für Implementierungen freigeben ist. Vielmehr soll der fließende Charakter der Spezifikation hervorgehoben werden:

> *The WHATWG specification is a continuously maintained living standard, with maturity managed at a very granular per-section scale, [...] this is intended to model the way in which specifications are approached in practice by implementors and authors alike. The W3C specification follows a more traditional style, with versioned releases [...], and with maturity management [...] the W3C specification has a version number (currently "5") and necessarily goes through periods of "feature freeze" where new features are not added, so that the specification can as a whole reach a more mature state.*

Browserhersteller und Entwickler von Web-Applikationen werden daher ermutigt, neben den veröffentlichten, versionierten Fassungen der Spezifikation auch neue Funktionen der fließenden Spezifikation zu implementieren. Diese orientieren sich an Angebot und Nachfrage und können weitaus schneller ergänzt und geändert sowie zum Status eines Standards erhoben werden.

Während sich das HTML-Format [HTML1] Anfangs noch auf reine Textinhalte beschränkte, wurde jede neue Version um immer mehr Möglichkeiten zur Interaktion (z.B. Formulare in Version 2 [HTML2]) und Gestaltung (z.B. Stylesheets in Version 4 [HTML4]) erweitert. HTML5 [HTML5] setzt hier als logische Fortsetzung an und führt viele neue Elemente, insbesondere in Hinsicht auf multimediale, reiche Inhalte, ein. So ist die native Wiedergabe von Audio- und Videoinhalten (`audio`- und `video`-Element) ebenso vorgesehen wie ein abgesteckter Bereich für 2D- und sogar 3D-Zeichenoperationen (`canvas`-Element). Daneben wurden allerdings auch viele rein strukturelle Elemente definiert, welche bisher semantisch nur andeutungsweise repräsentiert werden konnten (Kopf- und Fußbereiche, Teilabschnitte) oder nur programmiertechnisch möglich waren (erweiterte Formularelemente).

Eine weitere Neuerung ist die Definition eines konkreten Parsing-Algorithmus' für HTML5-konforme Browser[3]. Damit soll dem Missstand Einhalt geboten werden, dass es bisher jedem Browser selbst überlassen war, wie er ein HTML-Dokument verarbeiten soll. Insbesondere hinsichtlich fehlerhafter Dokumente wie z.B. falsch verschachtelter Elemente kommt es so nicht selten zu unterschiedlichen Ergebnissen abhängig davon, wie die jeweiligen Hersteller eine solche Situation bewerteten und behandeln. Mit dem ausführlich dokumentierten Algorithmus soll dem entgegen gewirkt und Browserhersteller motiviert werden, diesen zu implementieren. Es wird hier ein starkes Bestreben deutlich, Unterschiede zwischen Browsern abgesehen von der Einführung neuer Features zu verringern und damit den Entwicklungsprozess für neue Webseiten und -applikationen zu vereinfachen.

Generell war die Vereinfachung beim Verfassen HTML5-konformer Dokumente ein Hauptaugenmerk bei der Entwicklung des Standards. Dies zeigt sich bereits beim Aufbau

[3] http://www.w3.org/TR/2011/WD-html5-20110525/parsing.html (abgerufen am 02.06.2011)

eines minimalen Dokuments; ein Vergleich mit dem bisher üblichen Standard XHTML 1.0 [XHTML1]:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8"/>
    <title>Hallo Welt</title>
  </head>
  <body>
    <p>Hallo Welt
  </body>
</html>
```

Listing 1.1: Minimales HTML5-Dokument

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="de" xml:lang="de">
  <head>
    <meta http-equiv="content-type"
      content="application/xhtml+xml; charset=utf-8" />
    <title>Hallo Welt</title>
  </head>
  <body>
    <p>Hallo Welt</p>
  </body>
</html>
```

Listing 1.2: Minimales XHTML 1.0-Dokument

HTML5 unterscheidet sich von seinen Vorläufern allerdings dahingehend, dass mit dieser Spezifikation weitaus mehr als nur die strukturelle und semantische Komponente definiert wurde. So wurden auch einige JavaScript-Objekte standardisiert, welche bisher nur als Quasi-Standard allgegenwärtig waren, wie beispielsweise das `window`-Objekt. Auch interaktive Komponenten wie das Editieren von reichen Inhalten mit Hilfe des `contenteditable`-Attributs[4] und die Grundlagen für eine Interaktion mit Dokumenten per Drag-and-Drop (siehe Abschnitt 5.1) wurden klar spezifiziert. Gerade dieser Teil ist ohne die Hilfe einer Scriptsprache wie JavaScript allerdings wertlos, weshalb der Begriff HTML5 weitaus umfassender betrachtet werden muss.

1.2 JavaScript

Mit JavaScript steht im Web eine flexible und erweiterbare Scriptsprache zur Verfügung, deren Hauptaufgabe zumeist darin liegt, grundsätzlich statische HTML-Dokumente mit dynamischer Funktionalität zu versehen. Ursprünglich von Netscape während des Browserkriegs im Kampf gegen Microsofts Internet Explorer entwickelt [Schäfer11], wurde diese Sprache später in anderen Browsern adaptiert und deren Sprachkern schließlich sogar

[4] http://www.w3.org/TR/2011/WD-html5-20110525/editing.html#contenteditable (abgerufen am 02.06.2011)

in Form von ECMAScript [ECMA262-1] spezifiziert. Damit stand einer freien und offenen Weiterentwicklung nichts mehr im Wege.

Markant für JavaScript sind eine schwache Typisierung und dynamische Erweiterbarkeit. Dies macht sich auch im Entwicklungsprozess bemerkbar; eine klassische Vererbungshierarchie im Sinne von Klassen gibt es nicht, statt dessen kommt ein prototypisches Modell zur Erweiterung von Objekten zum Einsatz. Dieses hat den Vorteil, dass Methoden und Eigenschaften bei Kopplung an den Prototypen einer „Klasse“ nur einmal existieren, unabhängig davon, wie viel Instanzen es gibt. Zudem kann über den Prototypen zur Laufzeit jede daraus resultierenden Instanz mit neuen Eigenschaften und Methoden versehen werden.

Die Entwicklung professioneller Anwendungen im Web wird durch eine Vielzahl an JavaScript-Frameworks wie jQuery [JQuery] und MooTools [MooTools] erleichtert. Die durch JavaScript möglichen Operationen sind vielfältig: von der Manipulation von HTML-Dokumenten [DOM] über das Absenden von HTTP-Anfragen [XHR1] bis hin zum dynamischen Zuschalten von Funktionalität zur Verbesserung der Nutzererfahrung auf Webseiten. Während damit auch weniger nützliche Funktionalitäten implementierbar sind (endlose Popups, „Sperren“ des Kontextmenüs), entwickelte sich die Sprache in den letzten Jahren immer mehr zu einem essentiellen Bestandteil von Web-Applikationen.

Zu erwähnen ist in diesem Zuge auch der an die Objekt-Notation von JavaScript angelehnte Standard der JavaScript Object Notation [JSON], welcher heutzutage als kompakter Container für den Austausch von Daten immer häufiger einem XML-Dialekt vorgezogen wird [Almaer07].

1.3 Adobe Flex

Mit dem Flex SDK [Flex11] (aktuell in der Version 4.5) bietet Adobe [Adobe] ein Entwicklungs-Framework für den ebenfalls von Adobe entwickelten Flash-Player an. Für Desktop-Anwendungen kann Flex mit der Adobe Integrated Runtime [AIR] genutzt werden. Obwohl das Flex SDK eigenständig genutzt und damit erstellter Quellcode mit verschiedenen Werkzeugen in Anwendungen kompiliert werden kann, ist als zusammenhängende Entwicklungsumgebung der Adobe Flash Builder [FB] bzw. das passende Plugin für die Eclipse-IDE die erste Wahl.

Während mit dem Flash-Player schon lange eine Plattform für multimediale Inhalte und praktisch uneingeschränkter Gestaltungsfreiheit bestand, orientierten sich die dazu passenden Werkzeuge von Macromedia (später übernommen von Adobe) vorrangig an Grafikern und Webdesignern. Um die Plattform auch für Entwickler interessant zu machen, kündigte Macromedia am 17. November 2003 das Flex SDK an [Flex03]. Das Ziel war die Entwicklung von Rich Internet Applications (RIA). Schon von Anfang an bestand der Entwicklungsprozess darin, dass mit Hilfe einer XML-basierten Sprache (MXML) grafische

Benutzeroberflächen zusammengesetzt und durch die Verwendung von ActionScript mit Funktionalität versehen werden. Während dies auf den ersten Blick dem Modell von HTML und JavaScript zu ähneln scheint, ist jedoch anzumerken, dass MXML im Grunde nichts weiter als in XML formuliertes ActionScript ist. Sämtliche Applikationen lassen sich auch ohne MXML gleich entwickeln und tatsächlich wird beim Kompilieren von Flex-basierten Anwendungen als Zwischenschritt der MXML- in ActionScript-Code umgewandelt [MXML-AS11]. Während für die Kompilierung zu Anfang noch ein dedizierter Server erforderlich war, wurde der Prozess später in die Entwicklungsumgebung integriert.

Eine gravierende Änderung erfolgte mit Flex Version 2, da Adobe einen Großteil des SDK mit dieser Version unter einer Open-Source-Lizenz veröffentlichte[5]. Damit einher ging auch die Veröffentlichung von ActionScript in der Version 3.0, was sich verstärkt an objektorientierten Programmier-Paradigma orientiert [AS3].

Im Flex-SDK enthalten sind umfangreiche Pakete für den Aufbau reicher Benutzeroberflächen, inklusive vielfältiger Komponenten und Anpassung der Erscheinung über Stile und Skins sowie Effekte für die direkte Manipulation von Komponenten bzw. Übergänge zwischen Zuständen. Vielerlei Möglichkeiten für den Datenaustausch sind ebenso standardmäßig enthalten wie Klassen zur Manipulation von Datensammlungen.

1.4 CSS(3)

In den Anfangszeiten von HTML mehrten sich die Möglichkeiten der optischen Gestaltung von Dokumenten. Dafür waren jedoch immer Stil-Angaben direkt in der HTML-Struktur erforderlich, eine strikte Trennung zwischen Struktur und Erscheinung war nicht gegeben. Erst mit Version 4 von HTML erhielten die Cascading Style Sheets [CSS] mit ihrer ersten Version Einzug und begannen ihren Siegeszug.

Während in den ersten beiden Versionen ausschließlich statische Formatierungen mit der Erweiterung verschiedener Zustände möglich waren, führt Version 3 der CSS unter anderem Transformationen und Übergänge ein, welche bisher nur mit Hilfe von JavaScript möglich waren. Auch wurden sämtliche Erweiterungen im Vergleich zur aktuellen Version 2.1 modular gestaltet [CSSCurrent] und erwartungsgemäß unterschiedlich weit von Browserherstellern implementiert [CSSMozilla], [CSSIE], [CSSOpera], [CSSSafari].

Sowohl HTML als auch das Flex-Framework bieten die Möglichkeit, ihre Dokumente bzw. Anwendungen mit Hilfe von CSS zu formatieren. Das Flex-Framework beschränkt sich hierbei jedoch auf vergleichsweise einfache Selektoren[6] und eigens definierte Eigenschaften [FlexCSSProps]. Bis auf die Notation von Eigenschaften (Bindestrich- im Vergleich mit camelCase-Notation) gleicht sich die Syntax bei beiden Implementierungen jedoch.

[5] http://opensource.adobe.com/wiki/display/flexsdk/Legal+Stuff (abgerufen am 02.06.2011)

[6] http://opensource.adobe.com/wiki/display/flexsdk/CSS+Advanced+Selectors (abgerufen am 02.06.2011)

2 Konzeption des Prototypen einer Medienverwaltung

Im Zuge dieser Diplomarbeit ist eine zusammenhängende und in sich geschlossene Anwendung entwickelt worden, welche über eine einfache und intuitive Bedienung das Abrufen von Ressourcen praktisch beliebiger Quellen ermöglicht. Diese Ressourcen können betrachtet und in einfacher Form auch verwaltet werden. Ziel ist es, dieses Vorhaben unter Nutzung der zur Verfügung stehenden Techniken so umzusetzen, dass eine intuitive Nutzung ermöglicht wird. Der Prototyp sollte sich so verhalten, wie man es von anderen Programmen zur Verwaltung und Betrachtung von Dateien gewohnt ist und erwarten würde.

2.1 Grafische Bedienoberfläche und -konzept

Der Prototyp lehnt sich an übliche Werkzeuge zur lokalen Dateiverwaltung an und orientiert sich an diesen in Bezug auf die Bedienungskonzepte und der grafischen Gestaltung der Oberfläche. Die grundsätzliche Aufteilung enthält daher einen Bereich für hinzugefügte Quellen und eine Übersicht der von einer Quelle zur Verfügung gestellte Ressourcen:

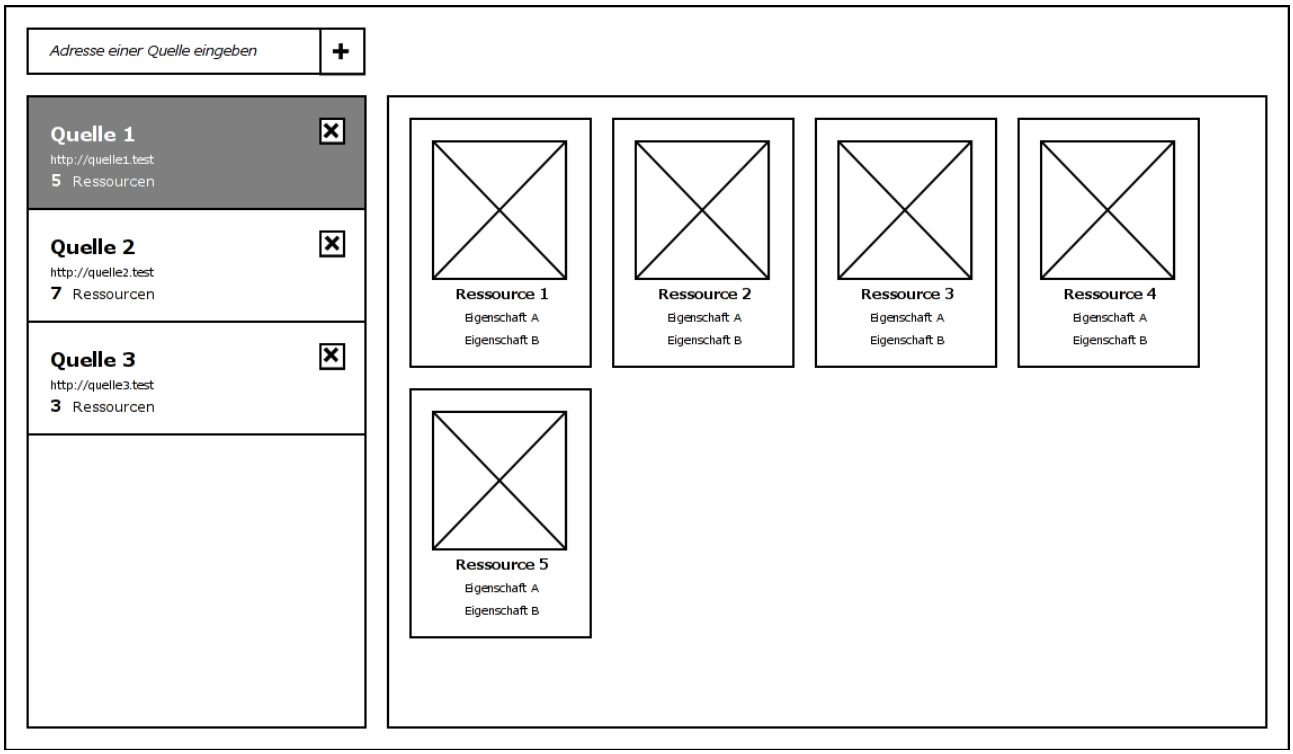

Abbildung 2.1: Schematischer Aufbau der grafischen Oberfläche

Ein Feld für die Eingabe der Adresse neuer Quellen gehört als erste Möglichkeit der Interaktion zur Anwendung. Nach Bestätigung wird versucht, grundsätzliche Informationen über die angegebene Adresse zu beziehen. Ist eine Verbindung nicht möglich oder entsprechen die empfangenen Daten nicht dem erwarteten Format wird eine entsprechende Fehlermeldung ausgegeben.

War das Abrufen der ersten Informationen erfolgreich, wird versucht, die Liste der über diese Quelle beziehbaren Ressourcen abzufragen. Gelingt dies nicht, wird eine entsprechende Fehlermeldung ausgegeben. In jedem Fall wird die Quelle im Bereich für hinzugefügte Quellen sichtbar. Über diesen Eintrag kann eine Quelle auch wieder entfernt werden.

Sind Ressourcen verfügbar, werden diese in einer für Dateiverwaltungen üblichen gitterartigen Anordnung im Bereich für Ressourcen angezeigt. Anzeige in diesem Sinne bedeutet hier die Darstellung einer Vorschau, sofern von der Quelle entsprechend geliefert sowie erster Metadaten wie Titel, Medientyp der Ressource und Datengröße. Ist keine Vorschau verfügbar, wird versucht, über den Medientyp ein geeignetes Symbol als Ersatzdarstellung zu finden. Diese entsprechen Grundtypen wie Audio, Bild, Text und Video. Die Anordnung der Ressourcen erfolgt über eine – möglichst konfigurierbare – Sortierung.

Bei grafischer Auswahl einer Ressource wird ein spezieller Betrachter über die Anwendung gelegt, welcher die eigentliche Ressource in größerem Format anzeigt. Die Information über den Ort der Ressource wurde zuvor beim Abruf der Ressourcen-Liste erhalten. Bei der Ansicht der Ressource werden verschiedene Anbieter für die Anzeige der Ressourcen unterschiedlichen Typs berücksichtigt. Diese Anbieter sind austausch- und erweiterbar; eine Anpassung für die Anzeige weiterer Medientypen ist damit einfach und flexibel möglich.

In der über den Betrachter verfügbaren Detailansicht besteht die Möglichkeit, in einen Vollbildmodus zu wechseln, um die jeweilige Ressource in bestmöglicher Ansicht zu präsentieren. Ebenso möglich ist ein Wechsel zur vorherigen und nächsten Ressource und schließlich das Ausblenden des Betrachters und damit die Rückkehr zur Standard-Ansicht.

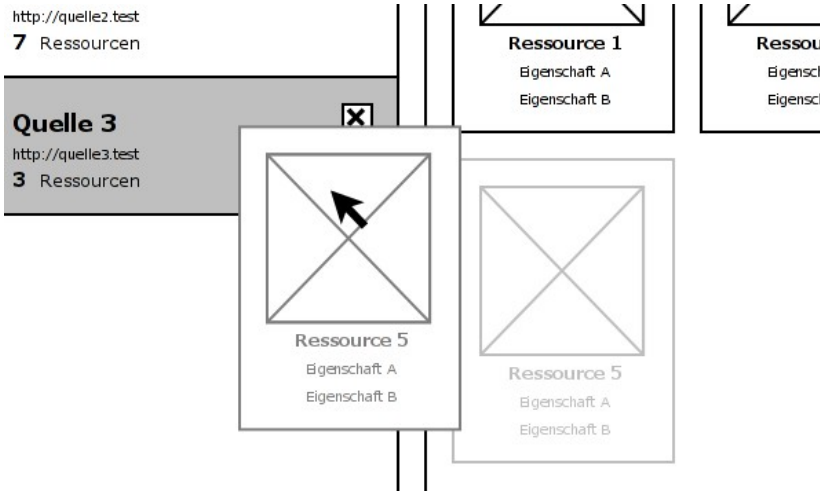

Abbildung 2.2: Schematische Darstellung einer Drag-and-Drop-Operation

Mit Hilfe von einfachen Drag-and-Drop-Operationen können eine oder mehrere Ressourcen von einer Quelle zu einer anderen verschoben oder kopiert werden. Dazu werden die beteiligten Quellen über die betreffende Anfrage informiert; der Abruf und die Speicherung obliegt den Quellen ebenso wie die Benachrichtigung über Erfolg oder Misserfolg der Operation. Ebenso über Drag-and-Drop ist das Hochladen von einer oder mehrerer Dateien

vom lokalen Speicher des Benutzers möglich. Dabei kann ein Nutzer Dateien direkt in das Fenster der Applikation ziehen und auf dem Eintrag einer Quelle bzw. dem Ressourcenbereich einer Quelle ablegen. Dadurch wird das Hochladen initiiert. Sowohl bei den Operationen von Quelle zu Quelle als auch vom lokalen Speicher zu Quellen erfolgt eine Information über den Fortschritt der betreffenden Operation und Benachrichtigungen über den Erfolg oder gegebenenfalls aufgetretene Fehler bei Misserfolg. Schließlich wird dem Nutzer auch die Möglichkeit geboten, Ressourcen per Drag-and-Drop aus dem Fenster der Applikation zum lokalen Speicher herunterzuladen.

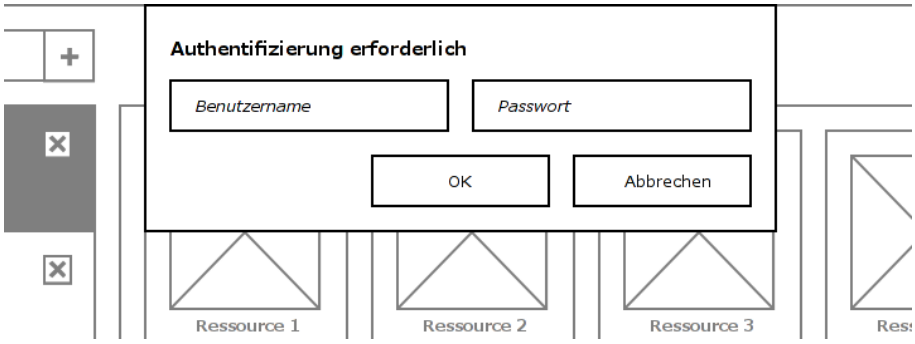

Abbildung 2.3: Schematische Darstellung eines Authentifizierungsdialogs

Beim Abrufen der Liste von Ressourcen einer Quelle, dem Kopieren und Verschieben, damit also dem Hinzufügen und Entfernen von Ressourcen und dem Hochladen von Dateien haben Quellen die Möglichkeit, Beschränkungen festzulegen. Diese äußern sich im einfachen Fall durch die Anforderung von Nutzerdaten zur Authentifizierung. Wird eine dieser beschränkten Operationen initiiert, wird dem Nutzer auf grafische Weise die Möglichkeit geboten, die erforderlichen Daten anzugeben. War die entsprechende Operation hiernach erfolgreich, werden die eingegebenen Authentifizierungsdaten für weitere Anfragen der gleichen Art zwischengespeichert.

Generell wird versucht, Präferenzen des Nutzers zu persistieren, so dass diese über mehrere Sitzungen hinweg beibehalten werden. Schließt also ein Nutzer den Browser, in dem die Anwendung aktiv ist, wird nach einem Wiederaufruf versucht, den letzten Zustand wiederherzustellen. Dies umfasst mindestens die Liste der bereits hinzugefügten Quellen und die davon zuletzt ausgewählte Quelle. Beim erneuten Aufruf wird damit versucht, wieder die grundsätzlichen Informationen der Quellen und ihre Liste der Ressourcen abzurufen.

2.2 Programmstruktur des Prototypen

Um die einzelnen konkreten Implementierungen der Anforderungen des Prototypen in den Technologien HTML5/JavaScript und Adobe Flex übersichtlicher vergleichen zu können, ist eine grundsätzliche Programmstruktur vorgegeben, an der sich beide Implementierungen orientieren. Dadurch ist auch zugleich der Aufwand für die beiden Implementierungen niedriger. Geringfügige Abweichungen liegen im akzeptierten Rahmen und obliegen aufgrund

der Eigenheiten der jeweiligen Technologien den konkreten Implementierungen. Im Folgenden werden die Zuständigkeiten der einzelnen Komponenten näher erläutert.

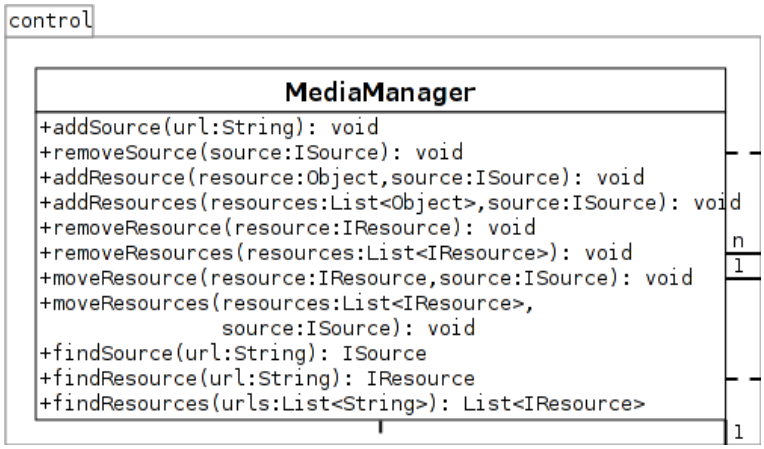

Abbildung 2.4: MediaManager-Klasse

Zentrale Instanz für sämtliche Operationen ist die Klasse `MediaManager`, welche die Liste der hinzugefügten Quellen (`sources`) in Form von Objekt-Instanzen umfasst, welche die Schnittstelle `ISource` implementieren. Der gleichen Schnittstelle entspricht eine Referenz auf die momentan ausgewählte Quelle (`selectedSource`). Der Zustand „ausgewählt“ hat hierbei hauptsächlich Relevanz für die grafische Oberfläche. Dies ist ebenso zutreffend auf die Referenz auf die momentan ausgewählte Ressource (`selectedResource`), welche die Schnittstelle `IResource` implementieren muss. Über die verschiedenen `add*`- `remove*`- und `find*`-Methoden können verwaltende Operationen an den abgerufenen Quellen und Ressourcen durchgeführt werden.

Zur Persistierung der Präferenzen des Nutzers kommt die Klasse `SettingsManager` zum Einsatz. Diese enthält einen privaten Verweis auf eine konkrete Implementierung zur Speicherung lokaler Daten. Über die Methoden `get` und `set` können Werte für Optionen abgerufen respektive festgelegt werden. Die Methode `setDefault` erlaubt es, einen Standardwert für Optionen festzulegen; dieser kommt dann zum Einsatz, wenn beim Abruf des Wertes einer Option zuvor noch kein Wert festgelegt wurde.

Zur Ausgabe von Informationen und zum allgemeinen Versenden von Benachrichtigungen werden diese unabhängig von jeglicher grafischer Darstellung in der Klasse `MessageManager` verwaltet. Diese Informationen werden durch Instanzen der Klasse `Message` repräsentiert, welche verschiedenen Typen für einfache Informationen (`INFO`), Fortschrittsanzeigen

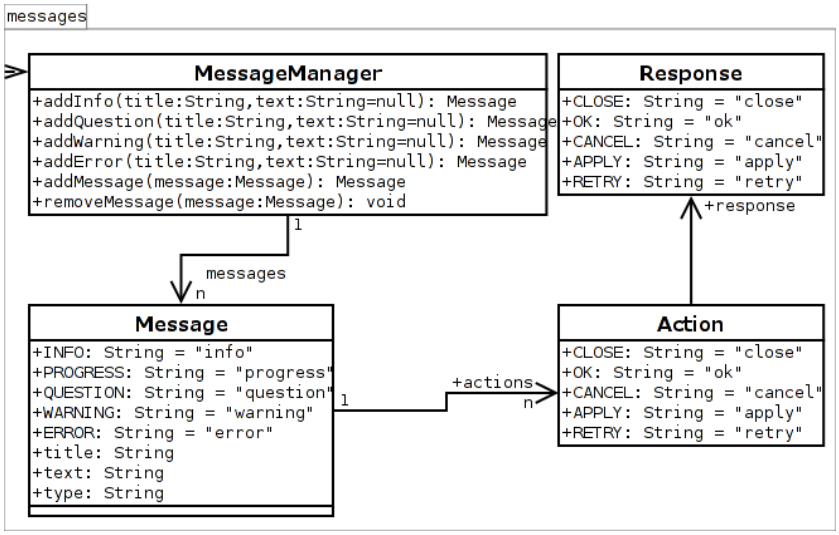

Abbildung 2.5: messages-Paket mit nachrichtenbezogenen Klassen

(PROGRESS), Fragen (QUESTION), behebbare Fehler in Form von Warnungen (WARNING) und Fehlermeldungen (ERROR) entsprechen. Zur einfachen Anwendung verfügt die Klasse MessageManager über Methoden zum bequemen Hinzufügen von Nachrichten dieser Typen. Nachrichten können mit Aktionen und den gewünschten Antworten (CLOSE zum Schließen, OK zum Bestätigen, CANCEL zum Abbrechen, APPLY zum Anwenden und RETRY zum Wiederholen von Aktionen im Fehlerfall) ausgestattet werden. Es obliegt der jeweiligen grafischen Anzeige, diese Aktionsmöglichkeiten darzustellen und die entsprechenden Antworten zu melden.

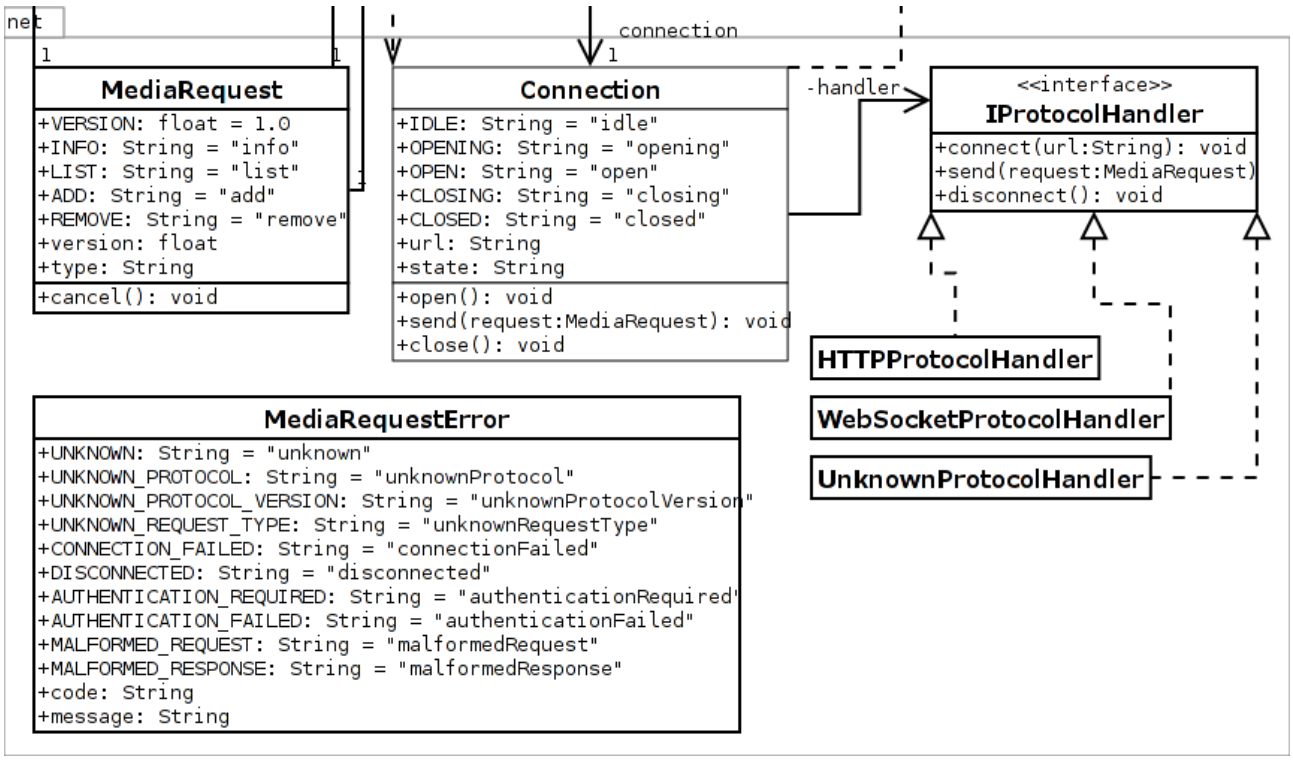

Abbildung 2.6: net-Paket mit netzwerkbezogenen Klassen

Beim Hinzufügen einer Quelle wird zuerst versucht, eine Verbindung herzustellen, welche ab diesem Zeitpunkt zur Kommunikation mit der Quelle dient. Dafür steht die Klasse **Connection** zur Verfügung. Diese bietet Methoden zum Öffnen (open) und Schließen (close) sowie zum Versenden von Anfragen (send) an und gibt Auskunft über ihren aktuellen Zustand (state), welcher initial im Leerlauf (IDLE) ist und über das Öffnen (OPENING) in den geöffneten (OPENED) bzw. über das Schließen (CLOSING) in den geschlossenen (CLOSED) Zustand übergeht.

Die Klasse Connection übernimmt hier allerdings nur eine kapselnde und abstrahierende Funktion. Die konkrete Kommunikation übernehmen die über die handler-Eigenschaft zugreifbaren Implementierungen der Schnittstelle IProtocolHandler, welche Methoden zum Verbinden (connect) und Trennen (disconnect) sowie zum Absenden von Anfragen (send) bereitstellen. Im Rahmen dieser Arbeit unbedingt erforderliche Implementierungen sind die Klasse HTTPProtocolHandler für die Kommunikation mit Quellen über HTTP sowie die Klasse WebSocketProtocolHandler für Quellen, welche für das WebSocket-Protokoll (siehe Abschnitt 4.2) zuständig sind. Zur transparenten Behandlung von unbekannten Protokollen stellt die Klasse

`UnknownProtocolHandler` eine besondere Implementierung der Schnittstelle `IProtocolHandler` bereit, welche ausschließlich Fehler meldet.

Um einen flexiblen und erweiterbaren Pool an Implementierungen für die Schnittstelle `IProtocolHandler` zu haben, greift die Klasse `Connection` und je nach Implementierung auch weitere Stellen im Prototypen auf die Klasse **`ProviderManager`** zu. Über diese können zur Laufzeit konkrete Anbieter für einen Dienst hinzugefügt bzw. entfernt (`addProvider`, `removeProvider`) und abgerufen (`getProviders`, `getProvider`) werden. Über die Art der Dienste wird keinerlei Aussage getroffen, da dies einzig dem verwendeten Kontext unterliegt. Die Klasse `ProviderManager` stellt somit nur ein unabhängiges, zentrales Register für Anbieter dar.

Für die Kommunikation zwischen den Klassen kommt ein ereignisbasiertes System zum Einsatz. Damit informiert z.B. die Klasse `MediaManager` über neu hinzugefügte oder entfernte Quellen bzw. Ressourcen und ermöglicht somit den grafischen Komponenten eine direkte Repräsentation der Änderungen. Auch bei der Kommunikation mit Quellen löst so die Klasse `Connection` Ereignisse aus, welche über den erfolgreichen oder fehlgeschlagenen Verbindungsaufbau, den Fortschritt während des Versands von Anfragen sowie die Antwort von Quellen informieren. Die Umsetzung dieses Systems obliegt der jeweiligen Implementierung und wird daher hier nicht explizit ausgeführt.

Die im Kontext der Kommunikation mit Quellen jedoch nur angedeuteten Anfragen sollen im Folgenden detaillierter erklärt werden, da hierfür eine festes Protokoll definiert wurde.

2.3 MediaRequest-Protokoll

Um eine nachhaltige und auch in Zukunft erweiterbare Kommunikation zwischen der Anwendung und Quellen zu gewährleisten, wurden Anfragen in Form des MediaRequest-Protokolls in Version 1.0 definiert. Damit wird ein konkretes Vokabular festgelegt, um es Anwendungen zu ermöglichen, grundsätzliche Informationen und Ressourcen von Quellen abzufragen sowie Dateien und Ressourcen zu Quellen hinzuzufügen und letztere von Quellen zu entfernen. Auf Seiten der Quelle bietet das Protokoll die Möglichkeit, über den Erfolg oder Misserfolg dieser Operationen zu informieren und die konkret angeforderten Informationen zu übermitteln.

Dabei beschränkt sich die Kommunikation nicht auf ein Anfrage-Antwort-Wechselspiel, da Quellen je nach Implementierung jederzeit eigenständig Informationen an die Anwendung versenden können.

Das Format der Nachrichten wird im Anhang formal per EBNF-Notation [EBNF] definiert. Der grundsätzliche Aufbau orientiert sich hierbei am JSON-Format. Das Format selbst ist jedoch nicht zwingend, lediglich die verwendeten Eigenschaften müssen in einer Anfrage unbedingt in der genannten Form enthalten sein. So ist bei der Kommunikation mit Quellen über HTTP alternativ die Formatierung gemäß RFC 2388 mit dem Typ „multipart/form-data“ vorstellbar

[RFC2388]. Gerade in Hinsicht auf die Übertragung von Binärdaten hochzuladender Dateien über die HTTP-POST-Methode ist diese Variante weniger platz- und zeitintensiv, da eine Kodierung der Binärdaten entfällt. Diese Entscheidung ist der betreffenden Implementierung überlassen.

Binärdaten von Dateien sind für die Übertragung mit dem Base64-Algorithmus [RFC4648] zu kodieren.

Passwörter werden in Version 1.0 des MediaRequest-Protokolls als SHA-1-Prüfsumme dargestellt und verarbeitet.

2.3.1 Grundbestandteil

Jede Anfrage und Antwort enthält grundsätzlich eine Information über die verwendete Protokoll-Version (`version`), derzeit 1.0. Eine eindeutige Kommunikation ist nur zwischen Anwendungen und Quellen mit der gleichen unterstützten Protokoll-Version sichergestellt. Zudem ist die Angabe zum Typ der Anfrage bzw. Antwort (`type`) Pflichtbestandteil. Hierdurch wird zum einen gewährleistet, dass auch eine asynchrone Kommunikation zwischen Anwendung und Quellen möglich ist, da Antworten auf Anfragen in beliebiger Reihenfolge gesendet werden können. Zum anderen können so jederzeit von Quellen ohne vorherige Anfrage Antworten gesendet werden. Hierdurch ist bspw. eine Benachrichtigung über anderweitig hinzugefügte Ressourcen möglich.

2.3.2 Anfrage-Typen

In dieser Version des MediaRequest-Protokolls werden vier verschiedene Anfrage- und Antworttypen definiert:

- Abfrage grundsätzlicher Informationen einer Quelle (`info`), dies umfasst den Titel sowie die Angabe, welchen Typ von Anfrage eine Quelle akzeptiert. Zusätzlich ist hier die Information zu finden, welcher Typ von Anfrage evtl. im Zugang beschränkt ist.
- Abfrage der von einer Quelle zur Verfügung gestellten Ressourcen (`list`), hier findet sich die absolute oder relative URL von Ressourcen sowie optional Metadaten wie Titel, Typ, Größe, usw.
- Anfrage zum Hinzufügen von Ressourcen und Dateien (`add`), das Abrufen der Daten einer Ressource obliegt der Zielquelle, Metadaten können hier mit übertragen werden. Bei Dateidaten sind die übertragenen Informationen auf Dateiname, -größe, -typ und die eigentlichen Daten beschränkt.
- Anfrage zum Entfernen von Ressourcen (`remove`), die Definition von Entfernen obliegt der Quelle. Für die Anwendung ist allein das Ergebnis der Operation entscheidend.

2.3.3 Anfragen

Anfragen verfügen zusätzlich über drei optionale Eigenschaften: eine Liste von Ressourcen (`resources`), eine Liste von Dateien (`files`) und Authentifizierungsdaten (`credentials`). Wie eine Quelle mit den Ressourcen und Dateien verfahren soll, entnimmt diese der Typ-Angabe der Anfrage. Schränkt eine Quelle einen Anfragetyp ein, kann die Anwendung vom Nutzer abgefragte Daten zur Authentifizierung übermitteln.

2.3.4 Antworten

Antworten verfügen neben den obligatorischen Angaben zur Protokoll-Version und Typ der Antwort über drei optionale Eigenschaften: eine Angabe von grundsätzlichen Informationen der Quelle (`info`), eine Liste von Ressourcen (`resources`) und Details für den Fehlerfall (`error`). Letztere umfassen neben vordefinierten Fehler-Codes (`code`) eine optionale textliche Information (`message`), hilfreich für den Fall, dass ein Fehler nicht mit den vordefinierten Fehler-Codes repräsentiert werden kann.

2.4 Die Quellen

Im Rahmen der Diplomarbeit sind vier verschiedene Quellen vordefiniert worden, welche mit unterschiedlichen Eigenschaften die Flexibilität des Prototypen unter Beweis stellen sollen. Diese Quellen können jederzeit über ihre zugehörige Adresse eingebunden und die dadurch verfügbaren Ressourcen abgerufen werden.

2.4.1 Ringo: HTTP-Quelle (PHP)

Diese Quelle erlaubt die Auflistung sowie das Hinzufügen neuer Ressourcen. Den basierend auf einer statischen Liste ausgelieferten Ressourcen fehlen sporadisch Eigenschaften wie Titel und Größe, die Prototypen müssen dies berücksichtigen. Das Hinzufügen ist zugangsbeschränkt und kann über den Nutzernamen „ringo“ und das gleichnamige Passwort „ringo“ nutzbar gemacht werden. Die Quelle lädt beim Hinzufügen von Ressourcen diese eigenständig von der Originalquelle und versucht auch evtl. Vorschaubilder zu übernehmen. Dateien werden über das `$_FILES`-Array verarbeitet.

2.4.2 Ichigo: HTTP-Quelle (PHP und MySQL)

Vergleichbar mit der Quelle Ringo erlaubt Ichigo das Auflisten von Ressourcen, jedoch nicht das Hinzufügen. Das Entfernen von Ressourcen ist jedoch möglich, die Dateien werden allerdings nicht physikalisch gelöscht sondern nur in der zugrunde liegenden Datenbank als solche vermerkt. Beim nächsten Abruf der Ressourcen-Liste werden so entfernte Ressourcen daher nicht mehr aufgeführt. Durch den Einsatz einer Datenbank wird die serverseitige Ressourcenverwaltung vereinfacht und dynamisch.

2.4.3 Mikan: HTTP-Quelle (PHP)

Als Quelle mit dem geringsten Funktionsumfang ist hier nur die Auflistung von Ressourcen möglich. Das markante an dieser Quelle ist jedoch, dass sie durch die Adresse `http://www.apps-entwickler.de/_brodala/mikan/` über eine gänzlich fremde Domain abgerufen wird. Der mögliche Zugriff über die Grenzen der Domain, in der sich der Prototyp befindet, wird hiermit unter Beweis gestellt. (Siehe Abschnitt 4.3)

2.4.4 Suika: WebSocket-Quelle (Python)

Diese Quelle ermöglicht als einzige eine Kommunikation über das WebSocket-Protokoll. (Siehe Abschnitt 4.2) Zum Einsatz kommt hierfür das auf dem Twisted-Framework von Python aufbauende txWebSocket [TxWebSocket]. Hiermit wird die gesamte Handshake-Logik des Websocket-Protokolls abstrahiert und die Umsetzung auf das Anlegen und Registrieren eines Handlers beschränkt. Dieser ist zuständig für die Kommunikation über das MediaRequest-Protokoll. Ein weiterer Handler ist für die spezifischen Anforderungen der Socket-Kommunikation in Flash implementiert. (Siehe Abschnitt 4.3.2) Neben dem Auflisten von Ressourcen ist auch das Hinzufügen mit Hilfe des Benutzernamens „suika“ und des Passworts „suika“ möglich.

Um eine der Stärken von WebSockets zu demonstrieren, sendet die WebSocket-Quelle 5 Sekunden nach dem Versand der Ressourcen-Liste eine unabhängige MediaRequest `add`-Antwort, um über eine neu hinzugefügte Ressource zu informieren. Hierdurch wird deutlich, dass per WebSocket eingebundene Quellen jederzeit solcherart Nachrichten versenden können. Ein praktischer Nutzen ist hier die Benachrichtigung über Änderungen ausgelöst durch Aktivitäten auf dem Server oder gar anderer Clients.

3 Umsetzung der grafischen Bedienoberfläche

Um die Fähigkeiten der beiden Implementierungen zu verdeutlichen und in einem konsistenten und zusammenhängenden Gewand zu präsentieren, ist die Erarbeitung einer dazu passenden grafischen Bedienoberfläche erforderlich. Diese nutzt die Möglichkeiten der optischen Gestaltung und erlaubt eine intuitive Bedienung.

Wie in Abschnitt 2.1 bereits erläutert, besteht das Ziel darin, eine Oberfläche vergleichbar mit üblichen Werkzeugen zur Dateiverwaltung zu erstellen. Die Quellen als Lieferanten für die zu verwaltenden und betrachtenden Ressourcen werden in Listenform angezeigt. Hinzugefügt werden Quellen über ein Eingabefeld, welches nach Möglichkeit eine automatische Vervollständigung der Standard-Quellen ermöglicht. Ein Eintrag in der Quell-Liste gibt über den Namen, die Adresse und die Menge der verfügbaren Ressourcen Auskunft. Über eine Schaltfläche können Quellen einfach wieder entfernt werden. Dadurch ist keine weitere Interaktion mit dieser Quelle möglich.

Bei Auswahl einer Quelle wird in einem größeren Detailbereich eine Übersicht der verfügbaren Ressourcen angezeigt. Per Maus kann mit diesen interagiert werden, was eine selektive Auswahl mittels `[Strg]`-Taste, eine Mehrfachauswahl mittels Selektionsrahmen und die Verschiebung von Ressourcen mittels Drag-and-Drop umfasst. Ebenfalls inbegriffen sowohl in der Quell- als auch der Ressourcen-Liste ist die Möglichkeit, Dateien vom lokalen Speicher des Nutzers mittels Drag-and-Drop abzulegen. Dadurch wird versucht, diese zur gewünschten Quelle hinzuzufügen.

Mittels Doppelklick auf eine Ressource wird ein Betrachter über die Anwendung gelegt, welcher die eigentliche Ressource in mittlerer Größe darstellt, um eine erste Ansicht zu ermöglichen. Bei Bedarf kann hier auch per Tastendruck in einen Vollbildmodus gewechselt werden, welcher den höchstmöglichen Detailgrad zugänglich macht. Der Betrachter ist auf eine Bedienung per Tastatur ausgelegt, weshalb ein Wechsel zwischen Ressourcen und das Schließen des Betrachters hiermit erfolgt.

Beim Hinzufügen und Entfernen von Ressourcen zwischen Quellen bzw. dem Hinzufügen von Dateien zu einer Quelle kann die Eingabe von Authentifizierungsdaten erforderlich sein. Die eigentliche Aktion wird dabei vorgemerkt aber erst nach Eingabe der Daten durchgeführt. Der hierfür modal über die Anwendung gelegte Eingabedialog fragt Nutzername und Passwort ab.

Ebenso über die Anwendung gelegt, werden in einem abgesteckten Bereich grafische Benachrichtigungen angezeigt. Diese umfassen bspw. die Information über hinzugefügte Quellen oder Ressourcen, Fehler beim Abrufen von Informationen über Quellen oder fehlgeschlagene Verbindungen sowie den Fortschritt beim Hochladen von lokalen Dateien zu Quellen. Mit diesen Benachrichtigungen kann über verschiedene Schaltflächen (z.B. Abbrechen, Wiederholen) interagiert werden. Das Schließen ist standardmäßig möglich.

3.1 HTML5/JavaScript

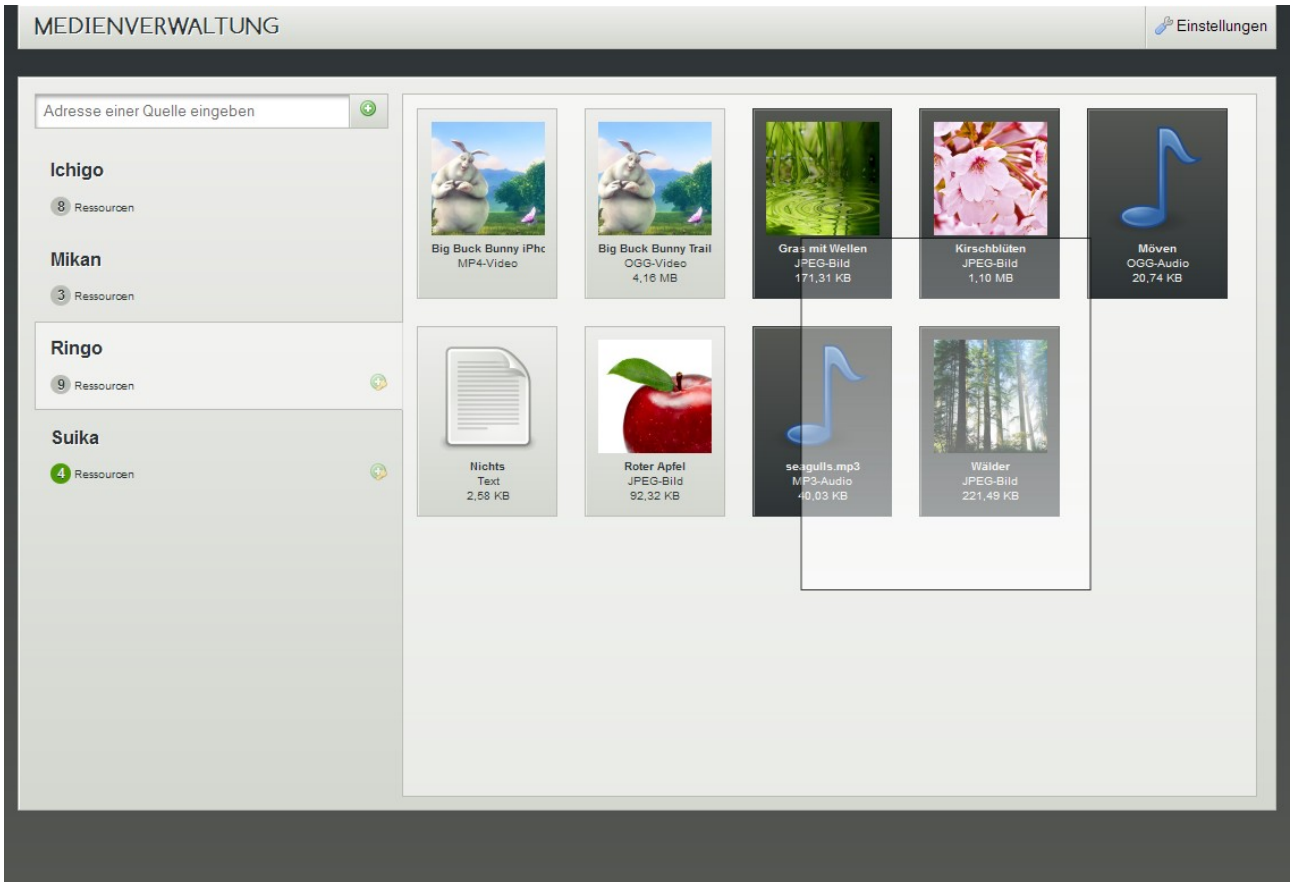

Abbildung 3.1: Medienverwaltung in HTML5/JavaScript

Die Umsetzung der grafischen Bedienoberfläche des Prototyps in HTML5/JavaScript bedient sich der durch HTML5 spezifizierten neuen Elemente zur besseren Strukturierung von Dokumenteninhalten. So ist z.B. der Kopfbereich der Anwendung mit Hilfe des `header`-Elements auch als solcher strukturell gekennzeichnet, der Inhaltsbereich ist über ein `section`-Element definiert. Auch wird der `data`-*-Attributraum zur programmunabhängigen Verankerung von Daten genutzt. Bemerkenswert ist auch die Fülle an neuen Typen für Formularelemente sowie sinnvolle Erweiterungen generell[7]. Neben Feldern für die Auswahl von Farben und Datum, Schieberegler und Schrittfelder für numerische Werte gibt es nun auch dedizierte Felder für die Eingabe von E-Mail- und Webadressen. Während kein Client gezwungen ist, diese in irgendeiner Form zu beachten, kann er diese jedoch als Orientierung verwenden, und das Eingabesystem daraufhin anpassen. Als Beispiel sei hier der Safari-Browser auf dem iPhone genannt [Nadel09]. Viel interessanter dagegen ist die automatische clientseitige Validierung von eingegebenen Werten. So kann der Versand eines Formulars schon vom Browser unterbunden werden, bis alle Felder den Vorgaben entsprechend ausgefüllt wurden. Kaum merklich und doch immer wieder per Script künstlich umgesetzt, sind Attribute für die Anzeige eines Platzhalter-Textes bei leeren Feldern (`placeholder`-

[7] http://www.w3.org/TR/2011/WD-html5-20110525/the-input-element.html (abgerufen am 18.06.2011)

Attribut), die Angabe eines bevorzugten automatisch fokussierten Feldes (`autofocus`-Attribut) und die Markierung von Pflichtfeldern (`required`-Attribut):

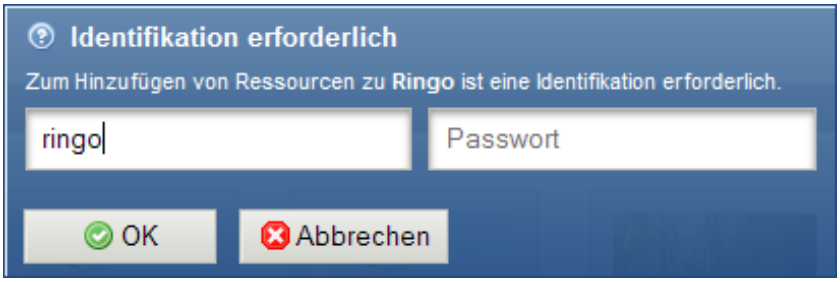

Abbildung 3.2: Authentifizierungsabfrage in HTML5 mit Platzhalter-Texten

Um auch in weniger modernen Browsern eine gute Standard-Ansicht auf HTML5-Dokumente zu haben, empfiehlt sich der Einsatz eines Reset-Stylesheets [ResetCSS]. Dadurch werden z.B. blockbildende Elemente auch optisch als solche gekennzeichnet. Zudem ist im Zusammenspiel mit dem Microsoft Internet Explorer wichtig, diesen immer in den Modus mit der bestmöglichen Darstellung zu versetzen. Hierfür hat Microsoft den Header `X-UA-Compatible` [XUAComp] eingeführt, wodurch man den zum Teil komplexen Algorithmus umgeht, mit welchem der Internet Explorer entscheidet, wie er ein Dokument rendern soll [IEDocMode].

Für den dynamischen Aufbau der Bedienelemente sowie zur Umsetzung des ereignisbasierten Nachrichtensystems kommt die Bibliothek jQuery zum Einsatz. Da HTML nach wie vor ein Format für Inhaltsdokumente ist, ist ein Konzept wie Widgets oder Komponenten nicht direkt vorgesehen. Statt dessen müssen solche händisch aufgebaut werden, können aber bei Auslagerung in eine Bibliothek flexibel wiederverwendet werden. Beispiele hierfür sind jQuery UI [JQueryUI] und YUI [YUI]. Um jedoch nicht zu viele Teile der Implementierung in externen Bibliotheken zu verstecken, wurde im Rahmen dieser Arbeit auf den Einsatz dieser Frameworks verzichtet und statt dessen eine eigenständige Entwicklung der Bedienelemente umgesetzt. Eine gute Strukturierung genügt allerdings noch nicht für den Aufbau ansprechender Bedienelemente, weshalb sich hierzu die Formatierung per CSS gesellt. Im Folgenden werden die im Rahmen des Prototyps entwickelten Bedienelemente und -konzepte aufgelistet.

Die Entwicklung eines Bedienelements umfasst somit die grundsätzliche Strukturierung per HTML, die Anwendung von Formatierung und ggf. sogar Animationen mittels CSS und die eigentliche Programmlogik in Form von JavaScript-Code. Eine Parallele zum MVC-Konzept ist unübersehbar und ermöglicht damit durch eine Entkoppelung dieser Teile eine leichtere Anpassung. So kann oftmals ohne Änderungen an Struktur und Programmcode die Erscheinung eines Bedienelements angepasst werden. Wichtig hierfür ist allerdings auch eine gute Struktur und sinnvolle Nutzung von Klassen zur Auszeichnung von Teilen des Bedienelements. (Z.B. der Rahmen, der Anfasser und die textliche Informationsanzeige einer Such- oder Fortschrittsleiste.)

Um den Komfort einer Mehrfachauswahl von Objekten vergleichbar mit einer Dateiverwaltung zu bieten, ist eine solche auch im HTML5/JavaScript-Prototyp der Medienverwaltung enthalten. Hierbei kommt das in HTML5 standardisierte `canvas`-Element [HTML5Canvas] zum Einsatz:

> *The canvas element provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, or other visual images on the fly.*

Bei dieser konkreten Umsetzung wird das Element absolut positioniert und in voller Größe über den Viewport gelegt, aber standardmäßig ausgeblendet. So bald ein `mousedown`-Ereignis in der Liste der Ressourcen registriert wird, wird der Inhalt des `canvas`-Elementes geleert, die Klickposition zwischengespeichert und das Element eingeblendet:

```
$selectionCanvas.attr({
                    width: $selectionCanvas.width(),
                    height: $selectionCanvas.height()
                  })
                  // Store the starting location
                  .data("startLocation", {x: e.pageX, y: e.pageY})
                  .show();
```

Listing 3.1: Einblendung des Auswahl-Rahmens in JavaScript

Die augenscheinlich sinnfreie Zuweisung der `canvas`-Dimensionen zu sich selbst dient dem Zweck des Zurücksetzens, wie in der Spezifikation beschrieben. Da nun das `canvas`-Element das gesamte Dokument überlappt, werden sämtliche Mausbewegungen an dieses geleitet. Dies wird genutzt, um bei jeder Bewegung einen Auswahlrahmen zu errechnen und diesen schließlich über die Methoden `fillRect()` und `strokeRect()` der `CanvasRenderingContext2D`-Schnittstelle [Canvas2D] entsprechend zu visualisieren. Dieser orientiert sich an der per jQuery bezogenen Position der Ressourcen-Liste relativ zum Viewport-Ursprung. Schließlich werden sämtliche Einträge in der Ressourcen-Liste gefiltert, indem für jedes ein positionsabhängiges Rechteck ermittelt und auf Überlappung mit dem Auswahlrechteck geprüft wird. Das Ergebnis dieser Filterung sind alle Einträge, die als ausgewählt markiert werden sollen. Die Erscheinung des Auswahlrechtecks ist praktisch frei definierbar, in der Implementierung kommt beispielhaft ein linearer Farbverlauf zum Einsatz, erzeugt über die Methode `createLinearGradient()` des Kontexts.

Als weiteres Beispiel des Einsatzes des `canvas`-Elementes ist ein Bedienelement zur visuellen Bewertung von Ressourcen implementiert. Hierbei hat sich seit langem eine Anordnung von fünf oder mehr grafischen Symbolen, meist Sternen, mit unterschiedlicher Färbung zur Verdeutlichung der ausgewählten Bewertung etabliert.

Beim Überfahren eines Sterns mit der Maus wird dieser sowie alle Sterne links von diesem mit Hilfe eines Schattens hervorgehoben, bei Klick wird die Bewertung festgelegt, womit die ausgewählte Bewertung visuell erhalten bleibt. Bei den Sternen selbst handelt es sich um eine

einzelne Bilddatei eines Sterns welche je nach Bewertung mehrfach gezeichnet wird:

Abbildung 3.3: Bedienelement für Bewertungen in HTML5

Das Bedienelement wurde so konzipiert, dass eine beliebige Grafikdatei als Visualisierung der Bewertung verwendet werden kann, die Grafikdaten für die inaktiven Sterne werden daher dynamisch ermittelt. Neben einer Verringerung des Alpha-Wertes kommt hier auch eine Luminanz-Formel zur Umwandlung der farbigen Ursprungsgrafik in ein Graustufen-Bild zum Einsatz [Lisci10]. Die Formel orientiert sich am Farbempfinden des menschlichen Auges. Die genutzte Methode `getImageData()` erlaubt das Abgreifen von Pixeldaten eines beliebigen Bereichs des `canvas`-Elements, die Methode `putImageData()` liefert das Pendant zum freien Platzieren von Pixeldaten:

```
function getInactiveImageData() {
  /* ... */
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.globalAlpha = 0.3;
  context.drawImage(ratingImage, 0, 0);

  var imageData = context.getImageData(0, 0, ratingImage.width, ratingImage.height);

  context.clearRect(0, 0, ratingImage.width, ratingImage.height);

  for (var i = 0; i < imageData.data.length; i += 4) {
    // Luminance formula by Marco Lisci
    var average = imageData.data[i + 0] * 0.30 + // R
                  imageData.data[i + 1] * 0.59 + // G
                  imageData.data[i + 2] * 0.11;  // B

    imageData.data[i + 0] = average;
    imageData.data[i + 1] = average;
    imageData.data[i + 2] = average;
  }

  return imageData;
}
```

Listing 3.2: Ermittlung inaktiver Bewertungsgrafiken in JavaScript

Die eigentliche Zeichenoperation für die Bewertungsgrafiken gestaltet sich überschaubar:

```
if (_highlighted) {
  context.shadowOffsetX = context.shadowOffsetY = 1;
  context.shadowBlur = 1;
  context.shadowColor = "black";
}
// Draw the active rating images
for (var i = 0; i < rating; ++i) {
  context.drawImage(ratingImage, i * ratingImage.width, 0);
}

if (_highlighted) {
  context.shadowOffsetX = context.shadowOffsetY = 0;
  context.shadowBlur = 0;
  context.shadowColor = null;
}
// Draw the inactive rating images
for (var i = rating; i < _maximum; ++i) {
  context.putImageData(inactiveImageData, i * ratingImage.width, 0);
}
```

Listing 3.3: Zeichnen der Bewertungsgrafiken in JavaScript

Dies sind nur einige für den Prototyp relevante Beispiele für den Einsatz des `canvas`-Elementes zum uneingeschränkten Zeichnen beliebiger Inhalte. Viele weitere Beispiele werden auf Seiten wie Canvas Demos zusammengetragen [CanvasDemos]. Neben der Nutzung für zweidimensionale Grafikinhalte ist auch mit einem verstärkten Interesse am 3D-Kontext des `canvas`-Elements zu rechnen. Dieser basiert auf der WebGL-Spezifikation, welche wiederum die OpenGL ES-Spezifikation als Grundlage hat [WebGL].

Eine wichtiger Hinweis ist, dass der Internet Explorer erst ab Version 9 [IE9Canvas] das `canvas`-Element kennt, daher älteren Versionen nativ keine dieser Zeichenoperationen ausführen können. Es gibt jedoch Lösungen, welche versuchen, dieses Defizit zu kaschieren. Die im Rahmen des Prototyps genutzte ist das Google-Projekt ExplorerCanvas, welche sich Microsofts VML, eine proprietäre Bibliothek und API für skalierbare Vektorgrafiken im Internet Explorer, zunutze macht [ExCanvas]. Nicht alle Funktionen des 2D-Renderkontexts des `canvas`-Elements lassen sich damit darstellen, aber für grundsätzliche Operationen ist diese Bibliothek eine angemessene Lösung.

Ein nicht zu verachtender Punkt bei der Betrachtung von Ressourcen ist die Möglichkeit, in einen Vollbildmodus zu wechseln. Programmatisch wird hierzu in der Klasse `Viewer` beim Druck auf die Taste `[F]` eine HTML-Klasse hinzugefügt bzw. entfernt. Die konkrete Umsetzung obliegt dann den Formatierungen des CSS, wobei diese sich auf das maximal mögliche beschränken:

```
#viewer.fullscreen {
  height: 100%;
  position: absolute;
  width: 100%;
}
```

Listing 3.4: Umsetzung des Vollbildmodus per CSS in HTML5/JavaScript

Hierbei handelt es sich erwartungsgemäß allerdings nicht um einen Vollbildmodus im Sinne der Definition. Der Betrachter nimmt lediglich den gesamten Platz des Viewports ein, ein Überspannen des Browserfensters oder gar des gesamten Bildschirms ist ausgeschlossen.

Dahin gehend gab es jedoch einen Vorstoß der Webkit-Entwickler, um genau dies zu ermöglichen [Almaer10]. Dieser Vorschlag beschränkte sich allerdings noch auf das HTML5 `video`-Element, weshalb in diesem Zuge über eine generische API nachgedacht wurde, welche es erlauben sollte, jedes Element in einem Dokument im Vollbildmodus darzustellen. Mozilla entwickelte daraufhin eine FullScreen API [FullscreenAPI], welche von Webkit schließlich übernommen und implementiert wurde[8], nutzbar ist diese vorerst allerdings ausschließlich in Apples Safari-Browser. Sicherheitsbedenken wurden geäußert, den Vollbildmodus eindeutig zu kennzeichnen und die Eingabe von Zeichen zu beschränken, um das Vortäuschen von Inhalten zu verhindern. Auch Nutzerinteraktion zum Wechsel soll unbedingt erforderlich sein. Weitere Entwicklungen und konkrete Implementierungen gilt es abzuwarten.

3.2 Adobe Flex

Jede grafische Applikation stützt sich in Adobe Flex auf einer Anordnung und Verschachtelung von MX- oder Spark-Komponenten. MX stellt hierbei grundsätzlich eine Menge von Komponenten dar, wobei viele diese Komponenten mittlerweile in der Spark-Architektur neu definiert wurden. Diese Architektur stellt einen Neuanfang der Konzeption dar und ermöglicht so eine Säuberung und bessere Kapselung von Funktionalität von Komponenten und ihrer Teile [Subramaniam10].

Zu dieser Kapselung zählt auch eines der Merkmale der Spark-Architektur: das veränderte Skinning. Skinning bezeichnet hier das Anlegen einer grafischen Komponente neben der eigentlichen Komponente selbst. Diese grafische Komponente wird schließlich in der Applikation gezeichnet und ermöglicht die Interaktion mit der zugrundeliegenden funktionellen Komponente. Eine klare Trennung von Erscheinung und Funktionalität war bei dieser Konzeption das Ziel.

Die in Adobe Flex standardmäßig nutzbaren Komponenten umfassen Container wie die eigentliche Applikation, Paneelen und Fenster. Daneben Grundelemente der Interaktion wie Schaltflächen und Eingabefelder sowie Komponenten für die Ausgabe von Daten. Für die Anordnung von Datensätzen stehen Listenbasierte Komponenten zur Verfügung, deren Erscheinung allerdings ebenso wie bei allen anderen Komponenten bei Bedarf gänzlich frei angepasst werden kann. Für die Darstellung der Ressourcen im Prototypen wird hier bspw. ein Layout für die kachelartige Anordnung genutzt.

[8] http://trac.webkit.org/changeset/75277 (abgerufen am 13.06.2011)

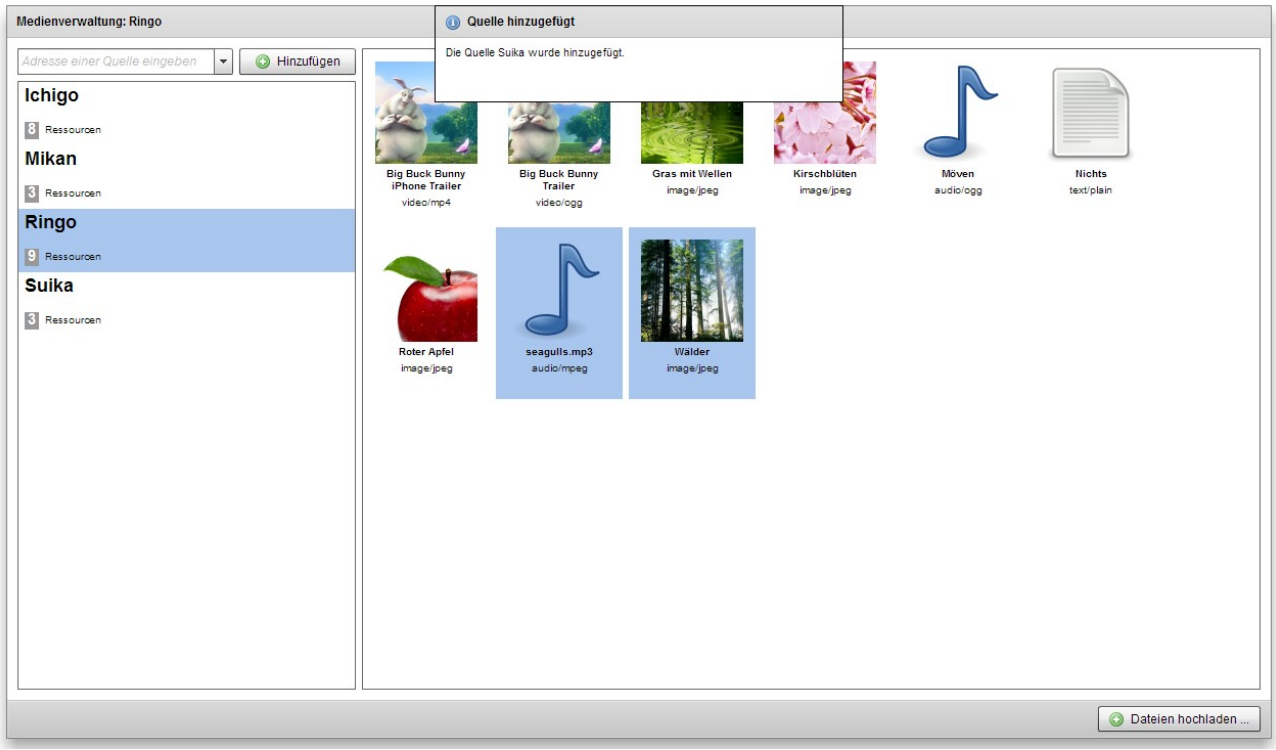

Abbildung 3.4: Medienverwaltung in Adobe Flex

Während sich HTML5 nur in Verbindung mit JavaScript um Bedienelemente erweitern lässt, gestaltet sich die Grundlage im Flex-Framework vollkommen anders. Hier liegt eine Umgebung vor, welche grundsätzlich für Applikationen und die darin verwendeten Bedienelementen, Komponenten genannt, konzipiert wurde.

Dies äußert sich auch bei der typischen Entwicklungsumgebung für Flex-Applikationen, dem Flash Builder von Adobe. Diese auf der Eclipse IDE basierende Entwicklungsumgebung vereint neben einer Code-Ansicht auch eine spezielle Designer-Ansicht und bietet somit die Möglichkeit, auf schnelle und intuitive Weise grafische Anwendungsoberflächen zusammenzustellen und zu veröffentlichen. Für einfache Anwendungen genügt hier sogar sehr wenig Anwendungslogik in Form von ActionScript-Code, da sich viele Vorgänge – wie die Manipulation von Datenquellen und die Animation von Bedienelementen – durch einfache Deklarationen und Datenbindungen vollziehen lassen.

Datenbindungen sind eines der heraus stechenden Merkmale von Flex und bedürfen einer näheren Erläuterung, da ein solches Programmkonzept in HTML5/JavaScript nicht verfügbar ist [FlexDataBinding]. Dabei wird mittels Datenbindungen auf einfache Art und Weise ein Objekt über einen Beobachter an eine Eigenschaft eines anderen Objektes gebunden:

```
<components:MessageList width="400" top="10" depth="1" horizontalCenter="0"
  dataProvider="{MessageManager.instance.messages}"/>
```

Listing 3.5: Einfache Datenbindung in MXML

Der Beobachter wird in MXML durch einfache Nutzung von Datenbindungen implizit und in

ActionScript über die Klassen des Pakets `mx.binding.utils`[9] explizit angelegt. Jedwede Änderung an dieser Eigenschaft registriert der Beobachter und leitet diese weiter. Bei Datensammlungen erstreckt sich beispielsweise dies über das Hinzufügen, Entfernen, Umsortieren und Aktualisieren einzelner Elemente. Die zu beobachtende Eigenschaft muss für eine erfolgreiche Datenbindung lediglich mit Hilfe des Metadaten-Tags `[Bindable]` als bindbar gekennzeichnet werden und die Regeln dieses Metadaten-Tags befolgen [FlexBindable]. Datenbindungen können durchaus auch komplexer gestaltet werden, da neben der zu beobachtenden Eigenschaft auch eine Verarbeitungsanweisung angegeben werden kann:

```
<fx:Script>
protected function uploadButtonEnabledDataProvider(source:ISource):Boolean {
  if (!source) { return false; }
  return MediaRequest.ADD in source.allowed;
}
</fx:Script>

<components:UploadButton height="25"
  enabled="{uploadButtonEnabledDataProvider(MediaManager.instance.selectedSource)}"
  filesSelected="onUploadButtonFilesSelected(event)"/>
```

Listing 3.6: Datenbindung mit Verarbeitungsanweisung in MXML

Komponenten selbst können durch die Manipulation ihrer Datenquelle Änderung über Datenbindungen publizieren. Nutzt mehr als eine Komponente eine Datenquelle, wirken sich Änderungen auf alle aus, evtl. angewandte Sortierungen und Filter werden so überall sichtbar.

Im Kontext listenbasierter Komponenten kommen auch immer wieder die verschiedenen Itemrenderer[10] zum Vorschein. Diese dienen der grafischen Darstellung von zugrunde liegenden Daten und werden für jeden Eintrag in einer Datenquelle wiederholt erzeugt. Während sich diese standardmäßig auf die einfache Anzeige der Textversion der zugrunde liegenden Einträge beschränkt, können ohne Probleme eigene Itemrenderer erstellt und uneingeschränkt gestaltet werden:

[9] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/mx/binding/utils/package-detail.html (abgerufen am 25.06.2011)

[10] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/supportClasses/ItemRenderer.html (abgerufen am 08.06.2011)

```
<components:MessagePanel id="message" width="100%" height="100%" minHeight="0"
  title="{data.title}" messageType="{data.type}">
  <components:CloseButton right="10" top="-23" includeIn="hovered,down"
    enabled="{data.type != Message.PROGRESS}" click="onCloseButtonClick(event)"/>
  <s:VGroup left="10" right="10" top="10"  bottom="10">
    <s:Label fontSize="11" text="{data.text}"/>
    <mx:ProgressBar id="progressBar" visible="{data.type == Message.PROGRESS}" label="%3%%"
      labelPlacement="right" source="{data}"/>
    <mx:HBox>
      <mx:Repeater id="actions" dataProvider="{data.actions}">
        <components:MessageActionButton messageAction="{actions.currentItem}"
          icon="{actionIconDataProvider(actions.currentItem)}"
          label="{actionLabelDataProvider(actions.currentItem)}"
          height="25" click="onActionButtonClick(event)"/>
      </mx:Repeater>
    </mx:HBox>
  </s:VGroup>
</components:MessagePanel>
```

Listing 3.7: MXML-Itemrenderer für Nachrichten

Das Resultat kann unter anderem wie folgt aussehen:

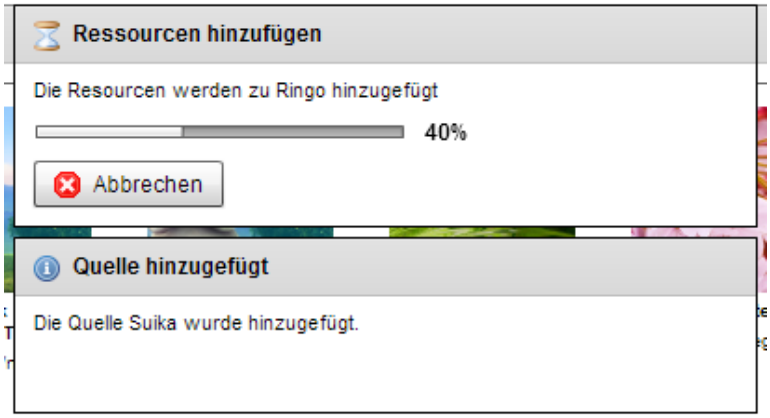

Abbildung 3.5: Nachrichtenliste der Medienverwaltung in Adobe Flex

Vergleichbar mit HTML5 erlaubt auch Flex das Formatieren von MXML-Komponenten mit Hilfe von Stylesheets [FlexCSS]. Hier wird die Kaskade als das C in CSS um die Vererbung von Formatierungen auf abgeleitete Komponenten erweitert. Wird also bspw. eine Formatierung für die `Group`-Komponente festgelegt, vererbt sich diese automatisch auf `HGroup`, `VGroup` und beliebige andere Ableitungen. Selektoren beschränken sich allerdings auf einfache Typen-, Vererbungs-, ID-, Klassen-, und Pseudoklassenselektoren; ausgefeilte Selektoren von CSS 2.1 und CSS 3 stehen daher nicht zur Verfügung. Neben allgemein gültigen Eigenschaften ist es MXML-Komponenten auch möglich, mit Hilfe des Metadaten-Tags `[Style]` neue Stileigenschaften zu definieren. Während Namensräume in HTML-Dokumenten mangels Bedarf noch eher ein Schattendasein fristen, sind sie im Kontext MXML gegeben durch die strenge Kapselung von Komponenten in Pakete Gang und Gäbe. Eine fein granulierte Selektion von gleichnamigen Komponenten abhängig von ihrem Ursprung ist somit problemlos möglich.

Während Formatierungen mittels CSS nur geringfügig Möglichkeiten zur optischen Anpassung

bieten, gehen die in Flex ebenfalls verfügbaren Skins [FlexSkins] einen großen Schritt weiter. Diese können über die `skinClass`-Eigenschaft der Basisklasse `SkinnableComponent` auf jede MXML-Komponente angewandt werden. Da es sich bei Skins selbst wiederum um Komponenten handelt, ist hier eine praktisch uneingeschränkte Gestaltung der so genannten Host-Komponente möglich. Ein Skin muss für die Anpassung von Komponententeilen diese lediglich über ihre IDs referenzieren. Diese werden als Skinteile bezeichnet. In der Flex-Implementierung im Rahmen dieser Diplomarbeit kommen Skins unter anderem bei der visuellen Gestaltung einer Wiedergabekomponente zum Einsatz.

Ein weiterer interessanter Aspekt bei der Arbeit mit MXML-Komponenten sind Zustände. Diese können von Komponenten definiert und basierend auf den definierten Zuständen Eigenschaften unterschiedliche Werte in Abhängigkeit vom aktuellen Zustand festgelegt werden. Auch können Komponenten explizit von Zuständen aus- bzw. eingeschlossen werden. So wird dies bspw. im Rahmen der Prototypen-Implementierung in Flex genutzt, um zusätzliche Informationen einer Quelle auf einfache Weise beim Überfahren des Eintrages in der Quell-Liste anzuzeigen:

```
<s:states>
  <s:State name="normal"/>
  <s:State name="hovered"/>
  <s:State name="down"/>
</s:states>

<s:HGroup>
  <s:Label text="{data.properties.title}"/>
  <components:CloseButton includeIn="hovered,down" click="onCloseButtonClick(event)"/>
</s:HGroup>
<s:Label text="{data.url}" visible="false" visible.hovered="true"/>
<s:HGroup>
  <s:Label text="{data.resources.length}"/>
  <s:Label text="Ressourcen" fontSize="10"/>
</s:HGroup>
```

Listing 3.8: Beeinflussung von MXML-Komponenten durch Zustände

Auch eine Gruppierung von Zuständen über ihre `stateGroups`-Eigenschaft ist möglich. Direkt im Zusammenhang mit Zuständen stehen Übergänge zwischen diesen. So steht eine Palette von vordefinierten Effekten zur Animation von Übergängen zur Verfügung, eigene können definiert werden. Effekte lösen wiederum selbst Ereignisse zu verschiedenen Zeitpunkten aus, welche bspw. zur Reaktion auf den Abschluss einer Animation genutzt werden können:

```
<s:transitions>
  <s:Transition fromState="inactive" toState="normal">
    <s:Resize target="{this}" duration="250" effectEnd="onShowEffectEnd(event)"/>
  </s:Transition>
  <s:Transition fromState="normal" toState="inactive">
    <s:Resize target="{this}" duration="250" effectEnd="onHideEffectEnd(event)"/>
  </s:Transition>
</s:transitions>
```

Diese Form von zustandsbewusstem Umsetzen von grafischen Oberflächen spart Zeit und Entwicklungsaufwand und bietet zugleich die Möglichkeit, die Erscheinung von Komponenten flexibel anzupassen.

Ein echter Vollbildmodus ist im Flash-Player und damit dem Flex-Framework seit jeher über das allgegenwärtige `Stage`-Objekt[11] und dessen `displayState`-Eigenschaft nutzbar. Eingaben vom Nutzer sind in diesem Modus in der Flash-Umgebung stark eingeschränkt, für die AIR-Umgebung gilt diese Beschränkung nicht. Auch dass der Wechsel in den Vollbildmodus als direkte Folge eines Mausklicks oder eines Tastendrucks erfolgen muss, ist in AIR nicht erforderlich. Die in der Flash-Umgebung im Vollbildmodus nutzbaren Tasten umfassen nicht-druckbare Zeichen inklusive der Cursortasten.

```
protected function onFullScreen(e:FullScreenEvent):void {
  if (e.fullScreen) {
    // Store state properties for restore upon leaving fullscreen mode
    this.beforeFullScreenInfo = {
      x: this.container.x, y: this.container.y,
      explicitWidth: this.container.explicitWidth, explicitHeight: this.container.explicitHeight,
      percentWidth: this.container.percentWidth, percentHeight: this.container.percentHeight
    };
    // Resize to stage fullscreen size
    this.container.setLayoutBoundsSize(this.stage.fullScreenWidth, this.stage.fullScreenHeight,
true);
    // Ensure dimensions are kept on layout changes
    // This can happen e.g. after the fullscreen info text ("Press Esc ...") was hidden
    this.container.explicitWidth = this.container.width;
    this.container.explicitHeight = this.container.height;
    // Move to 0,0
    this.container.setLayoutBoundsPosition(0, 0, true);
  } else {
    // Restore state properties from before entering fullscreen mode
    this.container.x = this.beforeFullScreenInfo.x;
    this.container.y = this.beforeFullScreenInfo.y;
    this.container.explicitWidth = this.beforeFullScreenInfo.explicitWidth;
    this.container.explicitHeight = this.beforeFullScreenInfo.explicitHeight;
    this.container.percentWidth = this.beforeFullScreenInfo.percentWidth;
    this.container.percentHeight = this.beforeFullScreenInfo.percentHeight;
  }
}
```

Listing 3.9: Anpassungen beim Vollbildmodus-Wechsel in Adobe Flex

Der `ResourceViewer` in der Flex-Implementierung des Prototypen sichert beim erfolgreichen Wechsel in den Vollbildmodus seinen derzeitigen Zustand bzgl. Positionierung und Dimensionen und vergrößert sich auf den gesamten auf der Bühne zur Verfügung stehenden Platz.

[11] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/display/Stage.html (abgerufen am 13.06.2011)

3.3 Einschätzung

Beim Aufbau grafischer Oberflächen halten sich HTML5/JavaScript und Adobe Flex die Waagschale. Während bei ersterem eine schnelle und unkomplizierte Entwicklung möglich ist, ist letzteres speziell für Applikationen konzipiert, was allein schon am Konzept der Komponenten erkennbar ist. Adobe Flex punktet allerdings insbesondere bei den Datenbindungen, welche in dieser Form nicht in HTML5/JavaScript existieren bzw. nur mit Umwegen und ansatzweise emulierbar sind. Das Zusammenspiel von Zuständen und Übergängen in Adobe Flex wäre ein großer Vorteil, wenn nicht das gleiche mit Hilfe von Klassen und CSS3 Transitions in HTML5 möglich wäre. Bei der Unterstützung des Vollbildmodus ist die weitere Entwicklung von HTML5/JavaScript abzuwarten; momentan gibt es dahingehend keine browserübergreifende Implementierung. In Adobe Flex ist dieser Modus dagegen standardmäßig verfügbar und ohne große Einschränkungen nutzbar.

Schließlich erleichtert der Flash Builder als Entwicklungsumgebung den Aufbau Flex-basierter Applikationen enorm, aber vergleichbare Anwendungen für HTML5/JavaScript werden nicht lange auf sich warten lassen, wie z.B. bereits an Hype [Hype] erkennbar ist. Zum Debugging in HTML5/JavaScript stehen z.B. mit Firebug [Firebug] und den standardmäßig ausgelieferten Entwicklungswerkzeugen von Google Chrome [ChromeDevTools] mächtige Werkzeuge zur Verfügung.

4 Einbindung von Quellen

Neben der Integration von Quellen, welche über HTTP bezogen werden können, werden als Alternative auch Quellen berücksichtigt, welche über Socket-basierte Verbindungen erreichbar sind. In diesem Zuge sind Objekte für diese beiden Protokolle entwickelt und in den Prototypen bei beiden Implementierungen integriert worden.

4.1 Quellen über HTTP

Zugriff auf Quellen, welche über HTTP abrufbar sind, wird von der Klasse `HTTPProtocolHandler` bereitgestellt. Dieser abstrahiert hierbei die Zustandslosigkeit von HTTP und erstellt für jede Anfrage ein neues Anfrage-Objekt. Aktuelle Informationen von Quellen sowie deren Ressourcen werden auf diesem Wege abgerufen und im Prototypen dargestellt. Dies ist in beiden Implementierungen gleich. Im Detail zeigen sich jedoch Unterschiede, auf welche im folgenden eingegangen wird.

4.1.1 HTML5/JavaScript

Die API Nummer 1 für HTTP-Zugriffe per JavaScript ist seit langem das `XMLHttpRequest`-Objekt [XHR1]. War es zu Beginn noch eine auf ActiveX basierte Eigenentwicklung von Microsoft, wurde es später adaptiert und standardisiert [Hopmann07]. Mit Version 2 [XHR2] dieses Objekts wurden einige sinnvolle Änderungen[12] vorgenommen:

> • *The ability to make cross-origin requests.*
>
> • *The ability to make anonymous requests — Referer, origin, and credentials are not part of the request.*
>
> • *The ability to register for progress events. Both for downloads (put listeners on the XMLHttpRequest object itself) and uploads (put listeners on the XMLHttpRequestUpload object, returned by the upload attribute).*
>
> • *The ability to override the media type and character encoding of the response through the overrideMimeType() method.*
>
> • *The ability to set a timeout for the request.*
>
> • *The ability to transfer Blob, File, and FormData objects.*
>
> • *The asBlob and responseBlob attributes for streaming large amounts of data to the client.*

Zudem wurde die Ereignisverwaltung mit dem neuen `XMLHttpRequest` Level 2 ausgefeilter gestaltet, indem einzelne Ereignisse in einer Schnittstelle namens `XMLHttpRequestEventTarget`

[12] http://www.w3.org/TR/2010/WD-XMLHttpRequest2-20100907/#differences (abgerufen am 18.06.2011)

untergebracht wurden. Verfolgte man früher Änderungen während einer Anfrage über den `onreadystatechange`-Handler und musste noch händisch die einzelnen `readyState`-Statusse sowie den schließlich empfangenen HTTP-Statuscode überwachen, stehen nun für alle relevanten Ereignisse bei Anfragen und Antworten entsprechende Handler zur Verfügung. Zudem gibt es eine gesonderte Eigenschaft, welche die gleichen Handler speziell für Uploads (`upload`) bereitstellt. Dies wurde in der HTML5/JavaScript-Implementierung berücksichtigt:

```
var requestObject = getRequestObject();

if (!requestObject) {

  $this.trigger(new MediaRequestError(request, MediaRequestError.CONNECTION_FAILED));
  return;
}

$(requestObject).bind({
  load: function(e) { /* Antwort verarbeiten, Ereignisverarbeitung beenden und Benachrichtigung
auslösen */},
  progress: function(e) { /* Ereignis weiterleiten */ },
  error: function(e) { /* Ereignisverarbeitung beenden und Benachrichtigung auslösen */ },
  timeout: function(e) { /* Ereignisverarbeitung beenden und Benachrichtigung auslösen */ }
});

$(requestObject.upload).bind({
  progress: function(e) { /* Ereignis weiterleiten */}
});
```

Listing 4.1: Anbindung an Ereignisse des XMLHttpRequest Level 2 Objekts

Die einzelnen Ereignisse werden verarbeitet und entweder direkt in der Aufrufhierarchie nach oben weitergeleitet (Beispiel: `progress`) oder umstrukturiert und weitergeleitet. So wird bspw. die Antwort einer Quelle mit Hilfe der JSON2-Bibliothek [JSON-JS] geparst und damit auf grundsätzlich syntaktische Korrektheit hin verifiziert. Ist dies erfolgreich, kann es sich hier prinzipiell um eine Antwort gemäß MediaRequest-Protokoll handeln, welche in den oberen Bereichen der Aufrufhierarchie weiterverarbeitet wird. Auf diese Weise wird der Zugriff auf Quellen über HTTP in der HTML5/JavaScript-Implementierung des Prototypen ermöglicht.

Beim Absenden von Anfragen nimmt hierbei das Objekt `FormData`[13] einen besonderen Stellenwert ein, da es hiermit erstmalig möglich ist, primitive Daten und Dateien mit einer einzelnen, gemeinsamen Anfrage zu übermitteln. Es entspricht daher dem Versand eines regulären HTML-Formulares, allerdings umgesetzt allein per JavaScript [MDCFormData]:

> *The FormData object lets you compile a set of key/value pairs to send using XMLHttpRequest. It's primarily intended for use in sending form data, but can be used independently from forms in order to transmit keyed data. The transmitted data is in the same format that the form's submit() method would use to send the data if the form's*

[13] http://www.w3.org/TR/2010/WD-XMLHttpRequest2-20100907/#the-formdata-interface (abgerufen am 18.06.2011)

> *encoding type were set to "multipart/form-data".*

Der Einsatz dieses Objektes bietet sich daher zur Kapselung von MediaRequest-Anfragen über HTTP in der HTML5/JavaScript-Implementierung an, da darin einfache textliche Informationen aber auch Dateidaten vorgesehen sind. Die Anwendung dieses Objekts, beispielhaft dargestellt anhand des Aufbaus von MediaRequest-Anfragen:

```
var formData = new FormData();

formData.append("mediaRequest[version]", request.version);
formData.append("mediaRequest[type]", request.type);

if (request.credentials) {
  formData.append("mediaRequest[credentials]",
    JSON.stringify(request.credentials));
}

switch (request.type) {
  case MediaRequest.ADD:
  case MediaRequest.REMOVE:
    if (request.resources) {
      formData.append("mediaRequest[resources]",
        JSON.stringify(request.resources));
    }
    // Intentionally not breaking here

  case MediaRequest.ADD:
    if (request.files) {
      request.files.forEach(function(file) {
        formData.append("mediaRequest[files][]", file);
      });
    }
    break;
}
```

Listing 4.2: Aufbau einer MediaRequest-Anfrage per FormData-Objekt

Äquivalent zur üblichen Verfahrensweise zum Hochladen multipler Dateien über reguläre HTML-Formulare wird eine implizite Liste an Dateien über die Syntax „`name[]`“ erzeugt. Dies erlaubt es bspw. PHP, den Array an Dateien entsprechend der POST-Anfrage auf- und vorzubereiten. Eine Variante mit einzeln benannten Dateien ist allerdings ebenso vorstellbar.

Die hier relevanten Verweise auf Dateien können aus offensichtlichen Sicherheitsgründen programmatisch nicht erzeugt werden sondern sind nur über Dateiauswahlfelder in Formularen sowie mittels Drag-and-Drop verfügbar. Bei beiden Vorgehen ist die Interaktion durch den Nutzer unerlässlich. Nähere Details hierzu finden sich in Abschnitt 5.1 .

Für Browser, die den Versand des `FormData`-Objekts über `XMLHttpRequest` noch nicht unterstützen, steht mit dem Projekt HTML5 FormData [HTML5FormData] in begrenzten Maßen eine Emulation zur Verfügung:

> *Emulate FormData object for browsers wich support FileAPI interface.*

Die genannte File API und die damit einher gehenden Anforderungen werden in Abschnitt 5.1 näher erläutert. Nicht explizit erwähnt wird allerdings die Abhängigkeit von der von Mozilla eingeführten und nicht-standardisierten Methode `sendAsBinary`[14]. Auch diese lässt sich in manch anderen Browsern emulieren[15], was aber wiederum weitere Abhängigkeiten mit sich bringt. Ein abschließender Überblick über die angesprochenen Objekte und ihre Eigenschaften:

Eigenschaft	`XMLHttpRequest`	`XMLHttpRequest Level 2`	`XDomainRequest`	`FormData`
Kapselung von Anfrage-Daten	✓	✓	✓	✓
Versand von Anfragen und Empfang von Antworten	✓	✓	✓	
Versand von Binärdaten		✓		
Benachrichtigung über Status	✓	✓	✓	
Benachrichtigung über Fortschritt		✓		
Verifikation domain-übergreifender Anfragen		✓	✓	
Einlesen von Dateidaten		✓		✓
Aufbereitung von Daten in HTTP-konforme Form				✓

Tabelle 4.1: Objekte zur HTTP-Kommunikation in JavaScript

Die erweiterte Funktionalität des Objekts `XMLHttpRequest` Level 2 ist hieraus ebenso ersichtlich wie die gänzlich neue Funktionalität durch das `FormData`-Objekt.

4.1.2 Adobe Flex

Die übliche Vorgehensweise zum Versenden von HTTP-Anfragen im Flex-Framework ist die Nutzung der `URLLoader`-Klasse[16]. Adobe beschreibt ihre Funktion wie folgt:

> *The URLLoader class downloads data from a URL as text, binary data, or URL-encoded variables. It is useful for downloading text files, XML, or other information to be used in a dynamic, data-driven application.*

Nicht unähnlich dem `XMLHttpRequest`-Objekt in JavaScript verfügt diese über Methoden zum Absenden (`load`) und Abbrechen (`close`) von HTTP-Anfragen sowie diverse Ereignisse zur Benachrichtigung über deren Fortschritt und Status. Hier inbegriffen sind Benachrichtigungen über das Ergebnis einer Anfrage sowie die über empfangene Antworten geladenen Daten.

Ein markanter Unterschied zeigt sich hier jedoch: während in Form des `XMLHttpRequest`-Objekts die zu versendende Anfrage und das versendende Objekt vereint sind, wurden diese beiden

[14] https://developer.mozilla.org/en/XmlHttpRequest#sendAsBinary() (abgerufen am 05.06.2011)

[15] http://javascript0.org/mediawiki/index.php?title=Portable_sendAsBinary&oldid=51(abgerufen am 05.06.2011)

[16] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/URLLoader.html (abgerufen am 05.06.2011)

Komponenten im Flex-Framework getrennt. Anfragen werden hierbei durch die Klasse `URLRequest`[17] repräsentiert. Dies hat den Vorteil, dass Instanzen dieser Klasse auch in anderen Kontexten wiederverwendet werden können, wie in der einleitenden Beschreibung der Klasse angedeutet wird:

> *URLRequest objects are passed to the load() methods of the Loader, URLStream, and URLLoader classes, and to other loading operations, to initiate URL downloads. They are also passed to the upload() and download() methods of the FileReference class.*

Mehrfach kopierter Code für die Bedienung der einzelnen Klassen kann somit vermieden werden. Auch ist hierdurch eine Kapselung der zugrunde liegenden Anwendungslogik möglich. Wie genau die zu versendenden Daten zusammengesetzt werden, ist für die genannten Klassen nicht relevant.

Das Hinzufügen von Daten zu `URLRequest`-Instanzen erfolgt über die `URLVariables`-Klasse[18]:

> *The URLVariables class allows you to transfer variables between an application and a server. Use URLVariables objects with methods of the URLLoader class, with the data property of the URLRequest class, and with flash.net package functions.*

Die prinzipielle Verwendung ähnelt dem `FormData`-Objekt der HTML5/JavaScript-Implementierung, das Anhängen von Dateien ist jedoch nicht vorgesehen. Das bedeutet, dass `FileReference`-Objekte (siehe Abschnitt 5.2) als Verweis auf lokale Dateien nicht an ein `URLVariables`-Objekt angehängt werden können, da dieses die Dateidaten nicht automatisch in eine für den Versand von HTTP-POST-Anfragen geeignete Form bringt. Ein einfaches Anhängen hätte zur Folge, dass schlicht die Zeichenkette „`[object FileReference]`“, übertragen würde, was der Textdarstellung der `FileReference`-Klasse entspricht. Die einzige Möglichkeit zur direkten Übertragung von Dateien besteht in den Methoden der `FileReference`-Klasse, beschränkt sich damit allerdings auf die Übertragung genau einer einzigen Datei.

Auf dieses Problem ist Mike Stead eingegangen und schuf die beiden Klassen `URLRequestBuilder` und `URLFileVariable`, welche die Erzeugung von `URLRequest`-Instanzen zur gemeinsamen Kapselung sowohl von primitiven als auch Dateidaten ermöglichen [Stead09]. Diese für den Prototypen dieser Arbeit prinzipiell hilfreiche Vorgehensweise zur Übertragung mehrerer Dateien über eine einzige Anfrage ist allerdings nur bis zum Flash Player einschließlich Version 9 nutzbar. Mit Version 10 des Flash Players wurden neue Restriktionen in Bezug auf von Nutzern initiierte Aktionen (UIA) eingeführt [Flash10-UIA]:

> *Among the changes implemented in Adobe Flash Player 10 is a number of user-initiated action (UIA) requirements to enhance overall security and also to cooperate with the emerging web security model as implemented by other web clients, such as browsers. […] When an ActionScript API function has a UIA requirement, that function can be called only*

[17] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/URLRequest.html (abgerufen am 05.06.2011)

[18] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/URLVariables.html (abgerufen am 05.06.0211)

> *in response to a user action such as a mouse click or key press. Some previously available ActionScript 2.0 and ActionScript 3.0 APIs have had UIA restrictions added in Flash Player 10.*

Die binären Inhalte von Dateien werden im Prototyp der Adobe Flex-Implementierung in Vorbereitung auf die Nutzung der Klasse `URLFileVariable` mit Hilfe der `FileReference`-Klasse[19] indirekt geladen. Aus diesem Grund greift hier genau diese Beschränkung. In dem Moment, indem eine solchermaßen aufbereitete Anfrage abgesendet werden soll, wird der Vorgang abgebrochen und folgende Fehlermeldung ausgegeben:

> *Error #2176: Certain actions, such as those that display a pop-up window, may only be invoked upon user interaction, for example by a mouse click or button press.*

Während der Hintergrund dieser Änderung nachvollziehbar und begründet ist, erschwert dies jedoch die saubere Trennung von grafischer Oberfläche (Auswahl der Dateien) und Datenverarbeitung (Lesen der Dateiinhalte). Ein Weg, dieses Sicherheitsfeature auf bedenklich einfache Weise zu umgehen besteht darin, gegen den Punkt 4.4 der RFC 2388 zu verstoßen und den Parameter „`filename`“ in etwas beliebig anderes umzubenennen [Warden08]. Adobe erläutert den genauen Zusammenhang hinsichtlich der Anwendung dieser Sicherheitsrichtlinie:

> *In Flash Player 10 and later, if you use a multipart Content-Type (for example "multipart/form-data") that contains an upload (indicated by a "filename" parameter in a "content-disposition" header within the POST body), the POST operation is subject to the security rules applied to uploads [...]*

Das Umgehen dieser Beschränkung ist daher nur möglich, weil der Programmcode der `URLLoader`-Klasse explizit auf den genannte Parameter in den POST-Daten prüft und Abwandlungen davon ignoriert. Der Quellcode dieser Klasse steht allerdings zur Verifikation nicht öffentlich zur Verfügung, da diese nicht zum Flex-SDK sondern zum geschlossenen Flash-Player gehört.

Mit diesem Vorgehen ist eine nahtlose Interaktion mit serverseitigen Technologien jedoch nicht mehr gewährleistet. So baut bspw. PHP seinen `$_FILES`-Array mit Hilfe von POST-Daten auf, die RFC 2388 entsprechen[20]. Benennt man den genannten Parameter um, schlägt die Erkennung fehl und die Daten landen im regulären `$_POST`-Array.

Ein manuelles Kodieren der Dateidaten mittels Base64, die Übertragung als reguläre Daten und die Dekodierung auf Seiten der Quellen ist daher hier unumgänglich. Eine komfortable Zusammenarbeit vergleichbar mit dem `XMLHttpRequest`-, dem `FormData`- und dem `File`-Objekt ist hier also nicht gegeben. Zur Verdeutlichung der Zusammenhänge auch hier eine Übersicht:

---

[19] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/FileReference.html (abgerufen am 05.06.2011)

[20] http://de2.php.net/manual/en/features.file-upload.post-method.php (abgerufen am 05.06.2011)

Eigenschaft	URLLoader	URLRequest	URLVariables	FileReference
Kapselung von Anfrage-Daten		✓		
Versand von Anfragen und Empfang von Antworten	✓			✓
Versand von Binärdaten				✓
Benachrichtigung über Status	✓			✓
Benachrichtigung über Fortschritt	✓			✓
Verifikation domain-übergreifender Anfragen	✓			✓
Einlesen von Dateidaten				✓
Aufbereitung von Daten in HTTP-konforme Form			✓	✓

Tabelle 4.2: Klassen zur HTTP-Kommunikation in Adobe Flex

Die auffällige Omnipotenz der `FileReference`-Klasse wird hieraus ebenso ersichtlich wie die gute Aufgabenteilung der URL*-Klassen. Ersteres lässt sich auf das Sicherheitsmodell in Adobe Flex zurückführen; alle möglichen auf Dateien ausführbaren Operationen sollen atomar und ohne weitere Klassen erfolgen. Dadurch wird die Angriffsfläche für Sicherheitslöcher verringert, da stets nur eine einzige Klasse involviert sein kann.

4.2 Quellen über WebSocket

Sockets dienen in der Computerwelt seit jeher zur Verbindung zweier Endpunkte zum bidirektionalen Datenaustausch. Eine Socket-Verbindung ist persistent, wobei jederzeit jeder der Endpunkte ohne vorausgehende Nachricht eine neue Nachricht an sein Gegenstück versenden kann. Auch kann jeder der Endpunkte bei Bedarf die Verbindung trennen. Die gesendeten Nachrichten unterliegen keinem bestimmten Format, können daher sehr kompakt und ohne unnötige Zusatzlasten (vgl. Header in HTTP) aufgebaut werden. Die konkrete Implementierung im Rahmen dieser Arbeit findet sich in der Klasse `WebSocketProtocolHandler`.

Mit dem durch HTML5 eingezogenen `WebSocket`-Protokoll [WSProtocol] und der passenden API [WSAPI] stehen die genannten Eigenschaften nun auch im Kontext Web zur Verfügung. Quellen, welche daher über dieses Protokoll abgerufen werden, können sich diese zunutze machen.

Ian Hickson definiert als Verfasser des `WebSocket`-Protokolls dessen Ziel hierbei wie folgt:

> *Conceptually, WebSocket is really just a layer on top of TCP that adds a Web "origin"-based security model for browsers; adds an addressing and subprotocol naming mechanism to support multiple services on one port and multiple host names on one IP address; layers a framing mechanism on top of TCP to get back to the IP packetmechanism that TCP is built on, but without length limits; and reimplements the closing handshake in-band. Other than that, it adds nothing. Basically it is intended to be*

> *as close to just exposing raw TCP to script as possible given the constraints of the Web.*

Zum Aufbau von WebSocket-Verbindungen wurde ein Konzept namens „Handshake“ definiert, welches sicherstellen soll, dass

1. zwei Endpunkte miteinander kommunizieren, die beide das WebSocket-Protokoll verstehen,
2. die Antwort auf eine Verbindungsanfrage wirklich von dem Endpunkt kommt, an den sie gerichtet war und
3. beide Endpunkte sich über das vom Anfragenden angeforderte Unterprotokoll einig sind.

Jegliche in dieser Arbeit genannte und implementierte Kommunikation über das WebSocket-Protokoll bezieht sich explizit auf die als hixie-76 bezeichnete Version des Handshake-Mechanismus’. Dies ist dem Umstand geschuldet, dass Webbrowser im Zeitraum der Implementierung des Prototypen nur mit dieser Version arbeiten konnten.

Am 26. November 2010 veröffentlichte Adam Barth auf der IETF-Mailingliste jedoch eine Nachricht[21], in welcher er auf ein gravierendes Problem mit dem Upgrade-basierten Handshake-Mechanismus bei Einsatz transparenter Proxy-Server hindeutet:

> *The core issue is that some number of transparent proxies do not understand the HTTP Upgrade mechanism and therefore don't understand that the remaining bytes sent by the attacker on the socket are not HTTP. These proxies treat these bytes as subsequent HTTP requests, letting the attacker either circumvent firewalls or, worse, poison the proxy's HTTP cache (depending on how the proxy is configured).*

Ein transparenter Proxy-Server ist hierbei ein Knoten in der Kommunikationskette, welcher üblicherweise nach dem Client-Zugang zum Internet platziert wird. Auszeichnend für diese Art Proxy-Server ist, dass sie keinerlei Veränderungen an Anfragen von und Antworten für Clients vornehmen und damit den Anforderungen des HTTP [RFC2616] folgen:

> *A "transparent proxy" is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification.*

Transparente Proxy-Server werden u.a. zur Anwendung von Sicherheitsrichtlinien in Firmen sowie Zwischenspeichern häufig angefragter Ressourcen genutzt.

Aufgrund des angesprochenen Problems entschlossen sich sowohl Mozilla [Blizzard10] als auch Opera [Ødegaard10] zwischenzeitlich die Unterstützung für das WebSocket-Protokoll und die -API in ihren Browsern gänzlich zu deaktivieren und nur per manueller Aktivierung einer Option (`network.websocket.enabled` in Firefox und `opera:config#UserPrefs|EnableWebSockets` in Opera) wieder zu reaktivieren.

[21] http://www.ietf.org/mail-archive/web/hybi/current/msg04744.html (abgerufen am 05.06.2011)

Das mit Version -76 (bei der IETF registriert als -00 [WSP-00]) veröffentlichte Problem wurde mit Folgeversionen des Protokolls behoben. Die aktuell veröffentlichte Protokollversion lautet -07 [WSP-07], welche in der Entwicklerversion von Firefox implementiert wurde[22] [Remi11]. Mit Version -03 wurde eines der Mankos des WebSocket-Protokolls beseitigt, indem Kompression mit dem DEFLATE-Algorithmus ermöglicht wurde. Die Übertragung von Binärdaten in Form von `Blob`-Objekten ist seit Version -01 auch möglich. `FormData`-Objekte werden jedoch nicht berücksichtigt, eine gemeinsame Übertragung von primitiven und Binärdaten ist daher nach wie vor nicht möglich.

Eine Übersicht über alle bisherigen Versionen des WebSocket-Protokolls und die damit einher gehenden Änderungen wird von den Entwicklern des pywebsocket-Moduls, einer WebSocket-Server-Implementierung in Python, geführt[23]. Dieses Modul ist eine vieler Möglichkeiten, eine serverseitige Implementierung des Websocket-Protokolls nutzen zu können.

4.2.1 HTML5/JavaScript

Die Nutzung der WebSocket-API in HTML5 gestaltet sich überschaubar und einfach. Ein minimales Beispiel für die Verbindung zu einem von websocket.org zur Verfügung gestellten WebSocket-Testdienst:

```
var socket = new WebSocket("ws://echo.websocket.org/")
socket.onopen = function() {
  alert("Connection opened, sending message"); this.send("Hello World!"); };
socket.onclose = function() { alert("Connection closed"); };
socket.onmessage = function(e) { alert("Got message: " + e.data); };
socket.onerror = function(e) { alert("Error: " + e); };
```

Listing 4.3: Aufbau einer WebSocket-Verbindung in JavaScript

Zum Zeitraum der Entwicklung des Prototypen im Rahmen dieser Arbeit stand in den Browsern nicht die aktuellste Fassung des WebSocket-Protokolls zur Verfügung, weshalb sich die Übertragung von Daten auf Text beschränkt. Eine dem MediaRequest-Protokoll entsprechende Nachricht wird daher in Form eines generischen JavaScript-Objekts vorbereitet und durch Serialisierung per JSON übertragen.

Hier inbegriffen ist das Laden von Dateidaten, was manuell über die `FileReader`-Klasse als Teil der HTML5 File API [FileAPI] erfolgt. Um einen Zusatzaufwand durch die Base64-Kodierung der binären Daten einzusparen, wird die `readAsDataURL`-Methode genutzt, welche dies automatisch übernimmt. Die potentiell aus mehr als einer Datei bestehende Liste der zu übertragenden Dateien wird abgearbeitet und nach dem erfolgreichen Laden aller Dateien die eigentliche Anfrage versendet:

[22] https://bugzilla.mozilla.org/show_bug.cgi?id=640003#c73 (abgerufen am 05.06.2011)

[23] http://code.google.com/p/pywebsocket/wiki/WebSocketProtocolSpec (abgerufen am 05.06.2011)

```
fileOperations.forEach(function(operation) {
  operation.reader.onload = function() {
    var operation = this.operation;
    delete this.operation; // Don't leak circular references
    fileOperations.splice(fileOperations.indexOf(operation), 1);

    request.files.push({
      name: operation.file.name,
      type: operation.file.type,
      size: operation.file.size,
      data: this.result.split(",")[1] // Drop "data:MIME/TYPE;base64"
    });

    // Finally submit data after the last operation has finished
    if (fileOperations.length == 0) {
      socket.send(JSON.stringify({mediaRequest: request}));
    }
  };
  operation.reader.onabort = function(e) { /* Remove operation from queue */};
  operation.reader.onerror = function(e) { /* Abort all operations, send notification event */ };
  operation.reader.onprogress = function(e) { /* Update request progress */ };
  operation.reader.readAsDataURL(operation.file);
});
```

Listing 4.4: Laden von Dateien vor Nachrichtenversand per WebSocket in JavaScript

Die Verarbeitung von Nachrichten, Fehlern und geschlossenen Verbindungen erfolgt äquivalent zur Implementierung des `HTTPProtocolHandlers`. Durch die Natur von Socket-basierten Verbindungen besteht aber jederzeit die Möglichkeit, dass eine Nachricht von der Quelle empfangen wird, ohne dass zuvor eine MediaRequest-Anfrage vorausgegangen ist.

Um das eventuelle Nichtvorhandensein der WebSocket-API in älteren Browsern zu kompensieren, steht mit dem Projekt web-socket-js [WS-JS] eine Emulation zur Verfügung. Diese erfolgt transparent und nur, wenn es keine native Implementierung der WebSocket-API gibt. Es wird hierbei ein Flash-Objekt in das HTML-Dokument eingebettet und über eine Wrapper-Klasse sämtliche Aufrufe an das eingebettete Flash-Objekt weitergeleitet. Verbindung und Kommunikation erfolgt hier daher über Flash-Sockets mit den damit einher gehenden Eigenarten und Einschränkungen. (Siehe Abschnitt 4.3.2)

4.2.2 Adobe Flex

Das im vorherigen Abschnitt erwähnte Projekt web-socket-js kann für die Implementierung des WebSocket-Protokolls in Adobe Flex sogleich wiederverwendet werden, da es in ActionScript verfasst ist und die für diese Diplomarbeit relevante Version des WebSocket-Handshakes implementiert. Die Integration erfolgt über die Verschiebung der Quelldateien aus dem Standardpaket von Flex in ein Paket namens `net.gimite`. Eine Schnittstelle namens `IWebSocket` gemäß der IDL der WebSocket-API wird definiert und die Klasse `net.gimite.WebSocket` entsprechend angepasst, um diese Schnittstelle zu implementieren. Dies umfasst das

Veröffentlichen der Eigenschaften `url`, `bufferedAmount` und `readyState` und die logische Deklaration der `readyState`-Werte als Konstanten. Von der Nutzung der Klasse `net.gimite.WebSocketMain` wird abgesehen, da diese – gekennzeichnet durch den Einsatz der `ExternalInterface`-Klasse – spezifisch für den Einsatz als eingebettetes Flash-Objekt aufgebaut wurde. Die Initialisierung eines WebSocket-Objektes erfolgt schließlich folgendermaßen:

```
this.socket = new WebSocket(
  new Date().time, url, null,
  (URLUtil.getProtocol(this.browserManager.base) + "://" +
URLUtil.getServerNameWithPort(this.browserManager.base)).toLowerCase(),
  null, 0,
  "", null,
  new DummyWebSocketLogger()
);

this.socket.addEventListener(WebSocketEvent.OPEN, this.onOpen);
this.socket.addEventListener(WebSocketEvent.MESSAGE, handler.onMessage);
this.socket.addEventListener(WebSocketEvent.ERROR, this.onError);
this.socket.addEventListener(WebSocketEvent.CLOSE, this.onClose);
```

Listing 4.5: Initialisierung eines WebSocket-Objekts in Flex

Zur Generierung des `Origin`-Headers bietet die Schnittstelle `IBrowserManager`[24] die Eigenschaft `base` an, welche über die `URLUtil`-KLasse[25] in ihre Bestandteile zerlegt werden kann. Der im Prototypen verwendete `DummyWebSocketLogger` dient nur der Befriedigung der Argument-Signatur und enthält sonst keinen Programmcode. Zustände und Ereignisse werden über die entsprechenden WebSocket-Ereignisse bearbeitet. Die Verarbeitung von Nachrichten und Fehlern erfolgt äquivalent zur HTML5/JavaScript-Implementierung.

Lediglich die Verarbeitung von Dateien beim Versenden einer MediaRequest-Anfrage zum Hinzufügen weicht geringfügig ab, da die `FileReference`-Klasse im Flex-Framework die Aufgaben der `File`- und `FileReader`-Klassen in HTML5/JavaScript in sich vereint:

[24] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/mx/managers/IBrowserManager.html (abgerufen am 20.06.2011)

[25] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/mx/utils/URLUtil.html (abgerufen am 20.06.2011)

```
request.files.forEach(function(file:FileReference, ...a):void {
  file.addEventListener(Event.COMPLETE, function(e:Event):void {
    var file:FileReference = e.target as FileReference;
    var type:String = file.type;
    var dotPosition:int = type.indexOf(".");

    if (dotPosition > -1) {
      type = MimeTypes.getMimeType(type.substr(dotPosition + 1));
    }

    mediaRequest.files.push({
      name: file.name,
      type: type,
      size: file.size,
      data: Base64.encodeByteArray(file.data)
    });

    if (request.files.every(isLoaded) && !requestCancelled) {
      handler.socket.send(JSON.encode({mediaRequest: mediaRequest}));
    }
  });
  file.addEventListener(IOErrorEvent.IO_ERROR, function(e:IOErrorEvent):void {
   /* Sent notification event */
  });
  file.addEventListener(ProgressEvent.PROGRESS, function(e:ProgressEvent):void {
    /* Update request progress */
  });
  // Start the file loading
  file.load();
});
```

Listing 4.6: Laden von Dateien vor Nachrichtenversand per WebSocket in Adobe Flex

Die Unterschiede zum `File`-Objekt in HTML5/JavaScript liegen hier darin, dass zum einen die über die `type`-Eigenschaft der `FileReference`-Klasse verfügbare Information tatsächlich nur die Dateierweiterung (z.B. `.png`) statt des nötigen MIME-Typen (z.B. `image/png`) enthält. Daher ist eine Erweiterung der `MimeTypeMap`-Klasse der as3corelib zum Einsatz gekommen, um dies zu korrigieren [AS3corelib]. Der MIME-Typ wird vielerorts verwendet, wie z.B. bei der automatischen Organisation von Dateien und Verzeichnissen auf den Quellen und zur Ermittlung eines geeigneten Symbols als Ersatz für Vorschaubilder.

Während in HTML5/JavaScript das `FileReader`-Objekt Zugriff auf Dateiinhalte ermöglicht, enthält bei der `FileReference`-Klasse die `data`-Eigenschaft nach dem Laden den Inhalt der Datei. Eine direkte Methode zum Einlesen als Base64-kodierte Zeichenkette ist nicht verfügbar, weshalb die Binärdaten manuell kodiert werden.

Da das Flex-SDK standardmäßig keine Möglichkeit bietet, Daten direkt ins JSON-Format zu konvertieren bzw. daraus auszulesen, findet sich auch hier für die as3corelib ein Einsatzort.

4.3 Domainübergreifende Zugriffe

Gegeben durch die Entwicklung des Webs verstärkte sich stetig der Ruf nach Möglichkeiten, Ressourcen über fremde Domains abrufen zu können. Dadurch soll sich die Möglichkeit bieten, Applikation im Web noch reichhaltiger in Bezug auf Inhalte umzusetzen, da viele interessante Inhalte erst durch fremde Quellen zusammengesetzt werden können. Bei einer Beschränkung auf Inhalte nur einer Domain ist dies nur in geringem Maße möglich oder erfordert eine Tunnelung beim Abruf fremder Inhalte, was zu Leistungseinbußen führt.

Das WebSocket-Protokoll speziell macht zu domainübergreifenden Anfragen keine Aussage, die Entscheidung, von wo Anfragen zugelassen werden, obliegt bei WebSocket-Verbindungen also gänzlich der Übereinkunft der beteiligten Kommunikationspartner. Diesen steht hierzu der `Origin`- bzw. `Sec-WebSocket-Origin`-Header zur Entscheidungsfindung zur Verfügung. Rein prinzipiell sind WebSocket-Verbindungen daher von überall zu jedem Server möglich.

4.3.1 HTML5/JavaScript

Während in HTML bei Ressourcen wie Bildern und Plugins schon immer eine domainübergreifende Einbettung möglich war, gestaltete sich dies bei Script-Inhalten schwieriger. So können zwar Scriptdateien selbst ebenso domainübergreifend eingebunden werden, aber darin befindlicher Code ist in seiner Laufzeit auf die aktuelle Domain beschränkt. Diese Vorgabe war lange Standard und unter dem Namen Same-Origin-Policy [SOP] bekannt.

Die eingangs im Prototypen verwendete Methode `getRequestObject` zum Zugriff auf Quellen über HTTP gestaltet sich wie folgt:

```
function getRequestObject() {

  var request = new XMLHttpRequest();

  if (!("withCredentials" in request)) { // Not XHR Level 2

    if (typeof XDomainRequest != "undefined") {

      request = new XDomainRequest();
    } else { // No CORS-compatible request object available

      request = null;
    }
  }

  return request;
}
```

Listing 4.7: Erstellung von JavaScript-Objekten für domainübergreifende Zugriffe

Hierbei kommt die empfohlene Methodik zum Einsatz, auf das Objekt `XMLHttpRequest` Level 2 zu

prüfen, indem die Existenz der `withCredentials`-Eigenschaft[26] überprüft wird [Ranganathan09]. Diese wurde erst mit Version 2 eingeführt und gibt an, ob bei domainübergreifenden Anfragen Cookies und Authentifizierungsdaten übertragen werden sollen.

Das Objekt `XMLHttpRequest` Level 2 ermöglicht zusammen mit der Spezifikation für Cross-Origin Resource Sharing [CORS] einen domainübergreifenden HTTP-Zugriff per JavaScript. Auf Seiten des Empfängers ist hierbei im einfachsten Fall nur ein weiterer HTTP-Header als Antwort auf solcherlei Anfragen zu senden: der `Access-Control-Allow-Origin`-Header. Dieser ermöglicht die Angabe, von welchen Ursprüngen (vom Anfragenden gesendet durch den `Origin`-Header) Anfragen erlaubt sind. Dabei sind ein, mehrere oder gar beliebige Ursprünge (mit Hilfe der Wildcard „*“) möglich[27]. Auch eine Portangabe ist hier vorgesehen. Der Browser übernimmt dann die Entscheidung, ob eine Anfrage schließlich übermittelt oder mit einer Sicherheits-Fehlermeldung abgewiesen werden soll.

Als Alternative zum `XMLHttpRequest`-Objekt für domainübergreifende Anfragen steht im Internet Explorer ab Version 8[28] das `XDomainRequest`-Objekt [XDR] zur Verfügung. Während dieses Objekt ähnliche Eigenschaften und Ereignisse anbietet, beschränkt sich die Auswahl an Daten, welche sich über die `send`-Methode übertragen lassen auf eine einfache Zeichenkette. Die von `XMLHttpRequest` Level 2 unterstützten Datentypen dagegen umfassen neben einfachen Zeichenketten Objekte vom Typ `Blob`, `Document` und `FormData`. Auch ein erweiterter Zugriff auf die gesendeten (`setRequestHeader`) und empfangenen (`getResponseHeader`) Header wie beim `XMLHttpRequest` Level 2 Objekt ist nicht vorgesehen.

Eine Übersicht über die üblichen Browser und ihre Unterstützung für domainübergreifende Zugriffe:

	 **Mozilla Firefox**	 **Microsoft Internet Explorer**	 **Google Chrome**	 **Opera**	 **Apple Safari**
Unterstützung ab	3.5[29]	8	~2.0[30]	-	4[31]

Tabelle 4.3: Unterstützung für domainübergreifende Zugriffe in aktuellen Browsern

Wie aus der Übersicht ersichtlich, ist die Unterstützung für domainübergreifende Zugriffe bereits weit fortgeschritten und kann daher bedenkenlos eingesetzt werden. Eine Ausnahme bildet hier der Opera-Browser, wo die weitere Entwicklung noch abzuwarten ist. Da der Marktanteil Operas allerdings vergleichsweise gering ist, 2,6% im April 2011 laut Statistik von w3schools.com [W3SchoolsStats], ist dieser Missstand verschmerzbar. Eine Besserung ist

[26] http://www.w3.org/TR/2010/WD-XMLHttpRequest2-20100907/#the-withcredentials-attribute (abgerufen am 18.06.2011)
[27] http://www.w3.org/TR/2010/WD-cors-20100727/#access-control-allow-origin-response-hea
[28] http://msdn.microsoft.com/en-us/library/cc287985(VS.85).aspx (abgerufen am 04.06.2011)
[29] https://developer.mozilla.org/En/HTTP_access_control (abgerufen am 05.06.2011)
[30] http://trac.webkit.org/browser/trunk/WebCore/xml/XMLHttpRequest.h?rev=42483 (abgerufen am 05.06.2011)
[31] http://developer.apple.com/library/safari/releasenotes/AppleApplications/rn-safari/_index.html (abgerufen am 18.06.2011)

absehbar, da Opera stets ein aktiver Teilnehmer bei der Verbreitung von Web-Standards ist [OperaWSC].

Eine Möglichkeit, domainübergreifende Zugriffe in mehr Browsern zu ermöglichen ist das Projekt `SWFHttpRequest`, welches Anfragen an eine eingebettete Flashdatei weiterleitet [SWFHR]. Diese Methode unterliegt allerdings den gleichen Beschränkungen wie die Implementierung des Prototypen dieser Diplomarbeit in Adobe Flex, auf welche im Folgenden eingegangen wird.

4.3.2 Adobe Flex

In Adobe Flex ist ein mit der CORS-Spezifikation in HTML5/JavaScript vergleichbares Verfahren vorhanden, um domainübergreifende Zugriffe zu ermöglichen. Auch hier muss das betreffende Ziel vorweg explizit sein Einverständnis erklären, Verbindungen anzunehmen. Während dies in HTML5/JavaScript über HTTP-Header geschieht, ist das Standardvorgehen bei Flash seit jeher eine so genannte Cross-Domain-Policy-Datei namens `crossdomain.xml`[32]. Über diese Datei, welche standardmäßig immer zuerst im Wurzelverzeichnis des Zielservers gesucht wird, kann über einige Richtlinien u.A. festgelegt werden:

- welche andere Policy-Dateien berücksichtigt werden sollen,
- von welchem Ursprung (Domain) Anfragen genehmigt werden,
- zu welchen Ports Anfragen gesendet werden dürfen,
- ob Anfragen von sicheren Ursprüngen erlaubt sind und
- ob zusätzliche HTTP-Header zulässig sind.

Dies übertrifft die einfache Vereinbarung, welche der HTML5 CORS-Spezifikation zugrunde liegt. Eine typische Policy-Datei, so im Rahmen dieser Arbeit zum Einsatz gekommen in Mikan, der über eine externe Domain erreichbaren Quelle:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-
policy.dtd">
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="master-only"/>
  <allow-access-from domain="dtm" />
  <allow-access-from domain="localhost" />
</cross-domain-policy>
```

Listing 4.8: Beispiel einer crossdomain.xml-Datei

Diese Datei legt fest, dass außer ihr keine anderen Policy-Dateien berücksichtigt werden sollen und gewährt Zugriffe von den beiden Domains „dtm“ und „localhost“. Sämtliche über HTTP z.B. in Form der `URLLoader`-Klasse getätigten Anfragen unterliegen der Bereitstellung dieser Datei, wenn es zu domainübergreifenden Anfragen kommt.

[32] http://www.adobe.com/devnet/flashplayer/articles/socket_policy_files.html (abgerufen am 08.06.2011)

Anders als im Kontext der WebSocket-API in HTML5/JavaScript gestaltet sich die Socket-Kommunikation in Adobe Flex durch die Flash-Player-Laufzeitumgebung. Denn auch hier wird beim Verbindungsaufbau jeder Art von Socket zuerst versucht, die besagte Policy-Datei abzurufen. Da es im Kontext Sockets keine Ressourcen gibt, wird hier allerdings nicht die zuvor genannte Datei abgerufen, sondern der Inhalt einer solchen Datei als allererste Antwort von Seiten des Kommunikationspartners erwartet. Zuvor fragt der Flash-Player die Datei durch das Senden einer Nachricht mit der Zeichenkette `<policy-file-request/>` an.

Das bedeutet, dass unabhängig von der konkreten Socket-Kommunikation stets eine Zeichenkette in Folge dieser ersten Nachricht gesendet werden muss, welche dem voran gegangenen Beispiel entspricht. Dazu ist anzumerken, dass der Flash Player immer zuerst versucht, diese über den Port 843 abzurufen und erst nach einer kurzen Verzögerung andere Ports in Betracht zieht.

Bei der WebSocket-Serverimplementierung im Rahmen dieser Diplomarbeit wurde dies berücksichtigt:

```
class FlashSocketPolicy(Protocol):
    def connectionMade(self):
        policy = '''<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.macromedia.com/xml/dtds/cross-domain-
policy.dtd">
<cross-domain-policy>
  <allow-access-from domain="*" to-ports="*" />
</cross-domain-policy>'''
        self.transport.write(policy)
        self.transport.loseConnection()

if __name__ == "__main__":
    from twisted.internet import reactor
    from twisted.internet.error import CannotListenError

    # ...

    # Run Flash socket policy file requests handler
    factory = Factory()
    factory.protocol = FlashSocketPolicy

    try:
        reactor.listenTCP(843, factory)
    except CannotListenError:
        logger.warning('Failed to start the Flash socket policy file '
                       'server on port 843; are you running as root?')
    else:
        # Move to unprivileged user
        import os
        os.seteuid(UID_WWW_DATA)

    reactor.run()
```

Listing 4.9: WebSocket-Server mit txWebSocket in Python

Hiermit ist es auch auf Flash-Sockets aufbauenden Implementierungen des WebSocket-Protokolls uneingeschränkt möglich, auf diese Quelle zuzugreifen.

4.4 Einschätzung

Der Zugriff auf Quellen über HTTP und das WebSocket-Protokoll ist sowohl mit HTML5/JavaScript als auch in Adobe Flex vergleichbar gut möglich. Mit XMLHttpRequest Level 2 ist die Funktionalität in HTML5/JavaScript nun mit Adobe Flex vergleichbar. Das unscheinbare aber sehr nützliche `FormData`-Objekt ist jedoch ein großer Vorteil von HTML5. Das dadurch sehr einfache gemeinsame Versenden von primitiven Daten, welche Zeichenketten und Zahlen umfassen, und mehreren Dateien ist so in Adobe Flex in neueren Flash-Playern nicht mehr möglich, da hier Flex wiederum zu den Vorgaben in HTML/JavaScript gleichgezogen ist. Ohne eine Interaktion durch den Nutzer sollten niemals Dateien einlesbar sein; beide Technologien stellen dies nunmehr gleichermaßen sicher.

Der generell schwierige Umgang mit Dateien in Adobe Flex bereitet einen zum Teil bemerkenswerten Mehraufwand ohne jedoch am Ende an die Funktionalität der Implementierung in HTML5/JavaScript heranzureichen. So gibt es Lösungen wie der besprochene `URLRequestBuilder`, allerdings ist dieser bedingt durch geänderte Sicherheitsrichtlinien nur bis zum Flash Player 9 nutzbar. Zudem besteht hier ein Problem im bereits angesprochenen Speicherbedarf. Dateien müssen vor dem Versand in ihrer Gänze in den Speicher geladen und für den Versand umkodiert werden. Beim `FormData`-Objekt von HTML5 dagegen ist es Browsern bspw. möglich, Dateien direkt als Stream und dadurch mit minimalem Speicherverbrauch zu übertragen. Ein Lösungsansatz wäre die Ausweitung der Funktionalität der `FileReference`- auf die `FileReferenceList`-Klasse. Da diese für die Auswahl mehrerer Dateien vorgesehen ist und zugleich aber auch die neuen Regeln zur Nutzerinteraktion befolgt, wäre sie hierfür sehr gut geeignet.

Im Umgang mit WebSocket-basierten Quellen ist lediglich die von Adobe Flex zwingende erforderliche Richtliniendatei negativ zu nennen; in HTML5/JavaScript gibt es diese Einschränkung nicht. Bei HTTP-Zugriffen ist die zwingende Erlaubnis domainübergreifender Zugriffe jedoch bei beiden Technologien vergleichbar und die Vorgehensweise zum Erteilen und Verarbeiten dieser Erlaubnis jeweils mit geringem Aufwand umsetzbar.

5 Drag-and-Drop von Ressourcen & Dateien

Um einen möglichst intuitiven Umgang mit Ressourcen (von Quellen bereitgestellt) und Dateien (vom Nutzer bereitgestellt) zu ermöglichen, liegt ein Hauptaugenmerk bei den Implementierungen des Prototypen auf dem Einsatz der Drag-and-Drop-Technik. Hierbei werden die entsprechenden Elemente per Mausklick angeklickt, gehalten und zum gewünschten Ziel gezogen. Bei Ablage wird die gewünschte Aktion durchgeführt. Das Ziel sollte vorweg oder während der Drag-Operation die Möglichkeit haben, die akzeptierten Datenformate bekanntzugeben.

5.1 HTML5/JavaScript

Das bisherige Vorgehen für Drag-and-Drop in HTML/JavaScript gestaltete sich oftmals wie folgt, das relevante Ereignis in Klammern [Ochard09]:

1. Klick auf das zu verschiebende Element (`mousedown`).
   a) Absolute Positionierung des Elements bzw. eines Klons davon. Ziel: Visualisierung der Drag-Operation.
   b) Ggf. Zwischenspeichern der Differenz zwischen Elementposition und Koordinaten des Maus-Ereignisses.
2. Start der Verschiebung (`mousemove`).
   a) Aktualisierung der Position des Elements bzw. dessen Klons basierend auf den aktuellen Mauskoordinaten. Gegebenenfalls Berücksichtigung der in Schritt 1b) zwischengespeicherten Differenz zum relativen Verschieben des Elements. Ziel: Vermeiden des Sprungs der linken oberen Ecke des Elements an die Spitze des Mauszeigers bei Verschiebung.
   b) Ermittlung der Position und Ausmaße möglicher Drop-Ziele im Dokument.
   c) Ermittlung der Elemente, die gerade vom verschobenen Element berührt werden.
   d) Markierung des möglichen Drop-Ziels bei Zutreffen, Entfernung der Markierung bei allen anderen Elementen.
3. Loslassen der Maustaste zum Durchführen der Drop-Operation (`mouseup`)
   a) Ggf. Entfernung des Klons und physikalische Verschiebung des Elements im DOM-Baum.

Während dieses Vorgehen gangbar ist, wird schon allein an den erforderlichen Schritten für die Positionierung des verschobenen Elements und der Ermittlung möglicher Drop-Ziele deutlich, dass es sich hierbei eher um eine Behelfslösung handelt. Der Grund für die aufwändige Ermittlung der potentiellen Ziele liegt darin, dass während einer Drag-Operation das verschobene Element stets alle anderen überdeckt und damit verhindert, dass diese ein `mouseover`-Ereignis auslösen können, womit die Erkennung vereinfacht würde. Der enorme

Mehraufwand bei vielen möglichen Drag-Elementen und Drop-Zielen in einem Dokument ist absehbar. Auch ist es Drop-Zielen nur über Umwege möglich, zu entscheiden, ob sie eine Drop-Operation akzeptieren oder abweisen. Eine weitere Einschränkung ist die Begrenzung auf das Dokument. Drop-Operationen mit Daten zu verarbeiten, welche von außerhalb des Browserfensters in dieses gezogen wurden (z.B. Dateien, selektierter Text in anderen Anwendungen) ist hiermit undenkbar. Hier bestand dringender Bedarf zur Abhilfe.

Schließlich wurden all diese Einschränkungen begutachtet und die Drag-and-Drop-API von HTML5 entwickelt [HTML5-DnD]. Grundvoraussetzung dafür, dass ein Element verschoben werden kann ist das in HTML5 definierte `draggable`-Attribut. Ausnahmen bilden die beiden Elemente `img` (generell) und `a` (bei Vorhandensein eines `href`-Attributs). Diese sind aus historischen Gründen standardmäßig verschiebbar, da die Grundlage für diese API bereits mit dem Internet Explorer 4 von Microsoft eingeführt und mit der Folgeversion weiterentwickelt wurde [MS-DnD]. Das theoretische Pendant zur Markierung von Drop-Zielen in Form des `dropzone`-Attributs wird momentan von Browsern noch nicht unterstützt, weshalb hier eine Zuarbeit mit JavaScript erforderlich ist. (Webkit-basierte Browser ausgenommen.[33])

Unumgänglich für die eigentliche Drag-and-Drop-Operation ist das Verarbeiten verschiedener Ereignisse für Quelle und Ziel. Das Vorgehen, im Vergleich zur vorherigen Auflistung:

1. Klick auf das zu verschiebende Element (im folgenden „Quelle“) und Start der Verschiebung (`dragstart`, `drag` während der Verschiebung)
   a) Festlegung der erlaubten Effekte über die `effectAllowed`-Eigenschaft: Verschieben (`move`), Kopieren (`copy`), Verlinken (`link`), Kopieren und Verschieben (`copyMove`), Kopieren und Verlinken (`copyLink`), Verlinken und Verschieben (`linkMove`), alle (`all`), keine (`none`)
   b) Festlegung eines Feedback-Bilds, wobei dies ein beliebiges Element sein kann (standardmäßig das Drag-Element, aber z.B. auch Bilder, andere Elemente oder gar per `canvas`-Element dynamisch gezeichnete Bilder)
   c) Befüllen des `DataTransfer`-Objekts mit den gewünschten Informationen in beliebigen Formaten
   d) Optional: Abbruch des Ereignisses, dadurch Abbruch der gesamten Operation
2. Verschiebung auf ein Ziel (`dragenter`, `dragover` während des Verschiebens über dem Ziel)
   a) Festlegung der erlaubten Effekte über die `dropEffect`-Eigenschaft: Kopieren, Verlinken, Verschieben und keine
   b) Entscheinung, ob eine Drop-Operation akzeptiert oder durch Abbruch des Ereignisses abgewiesen wird (kann auf Basis der Informationen im `DataTransfer`-Objekt erfolgen)
3. Optional: Verschieben des Elements aus dem Ziel (`dragleave`)
4. Loslassen der Maustaste zum Durchführen der Drop-Operation

[33] https://bugs.webkit.org/show_bug.cgi?id=58210 (abgerufen am 11.06.2011)

a) Auslesen der Daten im `DataTransfer`-Objekt (`drop` auf dem Ziel)

b) Optional: Entfernung des Elements von der Originalposition (`dragend` auf der Quelle)

Dieses umfangreiche Ensemble an Ereignissen ermöglicht eine fein regulierbare Festlegung, unter welchen Bedingungen von welcher Quelle zu welchem Ziel Drag-and-Drop-Operationen durchgeführt werden können. Da die schiere Menge an Ereignissen allerdings auch Unsicherheit hinsichtlich des Anwendungsbereichs erzeugt, folgt eine Übersicht über die Ereignisse:

Ereignis	Zuständigkeit	Ausgelöst auf
dragstart	Befüllen des `DataTransfer`-Objekts, Festlegung der erlaubten Effekte (`DataTransfer.effectAllowed`), Festlegung eines Feedback-Bilds (`DataTransfer.addElement()` oder `DataTransfer.setDragImage()`)	Quelle
drag	Aktualisierung der Drag-Visualisierung	Quelle
dragenter	Abweisung der Drop-Operation, muss manuell verhindert werden	Ziel
dragover	Festlegung des akzeptierten Effekts (`DataTransfer.dropEffect`)	Ziel
dragleave	Aktualisierung der Drop-Visualisierung beim Verlassen des Ziels	Ziel
drop	Auslesen des `DataTransfer`-Objekts	Ziel
dragend	Abschluss etwaiger Verschiebe-Aktionen durch Auslesen der `DataTransfer.dropEffect`-Eigenschaft	Quelle

Tabelle 5.1: Drag-and-Drop-Ereignisse in HTML5

(Anmerkung: in den aktuellen Versionen der implementierenden Browser muss die Drop-Operation durch Abbrechen des `dragover`-, nicht des `dragenter`-Ereignisses akzeptiert werden.)

In das `DataTransfer`-Objekt können neben den Standardformaten `text/uri-list` und `text/plain` beliebige Datenformate inklusive eigener festgelegt werden.

Da es sich bei diesen Ereignissen um reguläre Ereignisse oberster Ebene handelt, steigen diese vom auslösenden Element bis zur Dokumentenwurzel auf. Das bedeutet, dass bspw. die Bearbeitung des `dragstart`-Ereignisses einer Reihe von Elementen auf ihrem Eltern-Element erfolgen kann, auch wenn dieses selbst nicht als verschiebbar markiert wurde. So kann z.B. das `ul`-Element hierdurch `dragstart`-Ereignisse seiner `li`-Elemente verarbeiten. Werden neue `li`-Elemente hinzugefügt und als verschiebbar markiert, können diese automatisch vom `ul`-Element verarbeitet werden.

Bei den Effekten muss zwischen Quelle und Ziel eine Vereinbarung bestehen. Wurde bei der Verarbeitung des `dragstart`-Ereignisses bspw. als erlaubter Effekt das Kopieren festgelegt, muss das Ziel in seiner Verarbeitung des `dragover`-Ereignisses den gleichen Effekt festlegen. Dabei werden die kombinierten Bezeichner berücksichtigt. So wird eine Drag-Operation angenommen, wenn von der Quelle `copy` und vom Ziel `copyMove` festgelegt wurde. Die `dropEffect`-Eigenschaft gibt dann im `dragend`-Ereignis der Quelle Auskunft darüber, auf welchen Effekt sich

beide Parteien geeinigt haben. Kam es zu keiner Einigung, ist der Wert `none`.

Im einfachen Fall genügt das Setzen des `draggable`-Attributs, Befüllen des `DataTransfer`-Objekts im `dragstart`-Handler der Drag-Quelle, Abbrechen des `dragover`-Ereignisses auf dem Drop-Ziel sowie Auslesen des `DataTransfer`-Objekts im `drop`-Handler des Drop-Ziels.

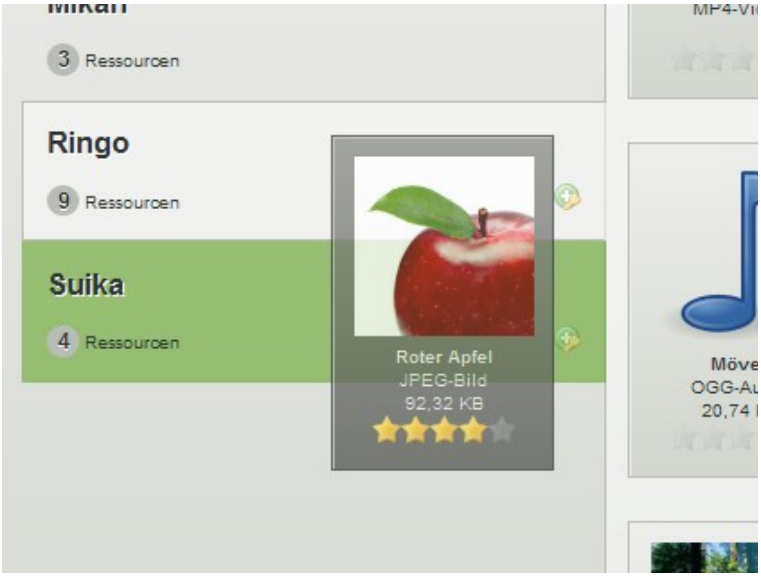

Abbildung 5.1: Mögliche Drop-Operation in HTML5

Bei der konkreten Implementierung im HTML5/JavaScript-Prototypen wird im dragover-Handler von Einträgen der Quell-Liste als auch der gesamten Ressourcen-Liste zuerst bestimmt, ob die betreffende Quelle das Hinzufügen von Ressourcen erlaubt. Danach wird geprüft, ob jede der hinzuzufügenden Ressourcen von einer anderen Quelle stammt. Sind beide Bedingungen erfüllt, wird die potentielle Drop-Operation zugelassen und entsprechend visualisiert:

```
function onDragOver(e) {
  var $item = $(this), targetSource = $item.data("source");
  // Indicate impossible action
  if (!targetSource || !(MediaRequest.ADD in targetSource.allowed)) { return true; }
  var urls = e.originalEvent.dataTransfer.getData("URL");
  if (urls) {
    urls = urls.split(",");
    for (var i = 0; i < urls.length; ++i) {
      var resource = manager.findResource(urls[i]);
      // Indicate impossible upload to the same source
      if (resource && (resource.source == targetSource)) {
        return true; // Stop and allow default action
      }
    }
  }
  $item.addClass("dragover");
  return false; // Cancel default action
}
```

Listing 5.1: dragover-Handler in JavaScript

Erneut zur Klarstellung: nur wenn diese Methode das Ereignis abbricht, wird die Drop-Operation zugelassen. Das Standardverhalten ist das Ablehnen der Operation, da andernfalls jedes Element im Dokument ein potentielles Ziel wäre.

Das `dragenter`-Ereignis wird in der Ressourcenliste nicht verarbeitet, in der Quell-Liste jedoch dafür genutzt, um nach einer Verzögerung auf die überfahrene Quelle zu wechseln. Dadurch ist es möglich, während einer Drag-and-Drop-Operation zu Ressourcenliste einer anderen Quelle zu wechseln. Dies dient lediglich der intuitiven Bedienung, da das Ergebnis einer Drop-Operation über einem Eintrag in der Quell-Liste und die zugehörige Ressourcen-Liste identisch ist.

Die Bearbeitung des `drop`-Ereignisses schließlich kann bedingt durch den Aufbau des Prototypen für Quell- und Ressourcenliste exakt gleich erfolgen, weshalb diese in Form der `DropTarget`-Klasse von beiden Listen gleichermaßen genutzt wird:

```
function onDrop(e) {
  var $target = $(e.target).removeClass("dragover");
  var targetSource = $target.data("source");
  if (!(MediaRequest.ADD in targetSource.allowed)) {
    return false;
  }
  var dataTransfer = e.originalEvent.dataTransfer;
  var files = dataTransfer.files;
  // Handle file upload requests (No such thing in IE yet)
  if (files && files.length > 0) {
    manager.addResources(files, targetSource);
  } else { // Handle internal drops
    var urls = dataTransfer.getData("URL");
    if (urls) {
      manager.moveResources(manager.findResources(urls.split(",")), targetSource);
    }
  }
  return false;
}
```

Listing 5.2: drop-Handler in JavaScript

Hierbei werden sowohl ggf. übertragene URLs von Ressourcen, als auch Dateien, abrufbar über die `files`-Eigenschaft des `DataTransfer`-Objekts, berücksichtigt.

Dadurch ist es möglich, vom Nutzer per Drag-and-Drop in den Browser gezogene Dateien zu verarbeiten. Bei der `files`-Eigenschaft handelt es sich hier um ein durch die HTML5 File API definiertes Array-ähnliches `FilesList`-Objekt. Ein darin enthaltenes `File`-Objekt verfügt über alle typischen Eigenschaften für den Zugriff auf Name, Größe und MIME-Typ einer Datei und erlaubt sogar über die Methode `slice()` eine neues Objekt zu erzeugen, welches einen Teil des Dateiinhalts, angegeben durch eine bytebasierte Anfangs- und Endposition, enthält. Ein lesender oder gar schreibender Zugriff auf diesen Inhalt ist damit allerdings nicht möglich. Das momentan einzige Anwendungsfeld für reine `File`-Objekte ist ein Hochladen von Dateien mit Hilfe von JavaScript. Durch Kürzen des Dateiinhalts lassen sich hier die zu übertragenden

Daten verringern. Ein einfaches Beispiel hierfür wäre die Übertragung der eigentlichen Audio-Daten einer MP3-Datei mit Verzicht auf die Metadaten in Form von ID3-Tags welche sich am Anfang bzw. Ende der Datei befinden [ID3]. Das `File`-Objekt erlaubt dies bereits clientseitig, andernfalls müssten alle Daten übertragen und die Entfernung der ID3-Tags serverseitig vorgenommen werden.

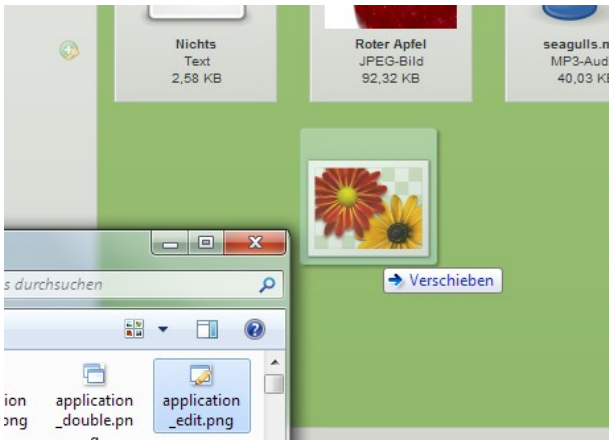

Abbildung 5.2: Mögliche Drop-Operation von Dateien in HTML5

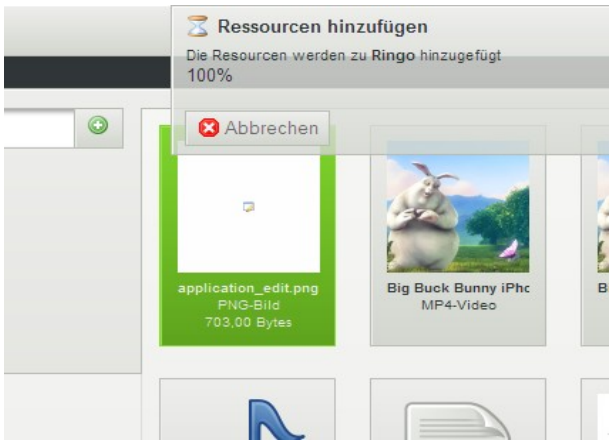

Abbildung 5.3: Anzeige hochgeladener Dateien in HTML5

Zu diesem Zwecke wurde das `FileReader`-Objekt konzipiert, welches das Auslesen des Dateiinhalts als Array-Puffer, binäre Zeichenkette, einfachen Text oder als Data-URL [RFC3797] erlaubt. Verschiedene Ereignisse informieren über den Zustand der Lese-Operation, wobei am Ende bei Erfolg über die `result`-Eigenschaft uneingeschränkter Zugriff auf die Dateidaten besteht. Hiermit ist bspw. eine direkte Anzeige durch Verwendung der Data-URL auf der Website ebenso denkbar wie die Umsetzung von Editoren, direkt verfügbar im Browser und ohne weiteren Bedarf an Plugins. Sollen Dateien allerdings ausschließlich zum Hochladen zu einem Server eingelesen werden, ist das `FormData`-Objekt vorzuziehen, da beim Einlesen großer Dateien entsprechend viel Speicherplatz verbraucht wird.

Sicherheitsbedenken können hier zerstreut werden, da es keine Möglichkeit gibt, ein `File`-Objekt ohne Zutun eines Nutzers zu erzeugen. `File`-Objekte werden einzig und allein per Drag-and-Drop bzw. über ein althergebrachtes Datei-Upload-Formularfeld verfügbar. Eine Ausnahme dieser Regel stellt die Writer-Erweiterung der File-API [FileWriterAPI] dar, welche momentan jedoch noch von keinem Browser unterstützt wird. Dadurch ist es Web-Applikationen möglich, selbst Dateien anzulegen, mit Inhalten zu befüllen und auf dem lokalen Speicher des Nutzers zu speichern. Das Sicherheitskonzept hier ist nach momentanem Stand noch unklar und kann erst bei ersten Implementierungen eingeschätzt werden.

5.2 Adobe Flex

Seit Version 3 des Flex-Frameworks steht für Drag-and-Drop-Operationen der `DragManager`[34] zur Verfügung. Die Beschreibung aus Adobes ActionScript-Referenz:

> *The DragManager class manages drag and drop operations, which let you move data from one place to another in a Flex application. For example, you can select an object, such as an item in a List control or a Flex control, such as an Image control, and then drag it over another component to add it to that component. […] All Flex components support drag and drop operations. Flex provides additional support for drag and drop to the List, Tree, and DataGrid controls.*

Das generelle Vorgehen für sämtliche MXML-Komponenten besteht darin, im `mouseDown`-Handler die Methode `DragManager.doDrag()` aufzurufen und damit die Drag-Operation zu initiieren. Die zu verschiebenden Daten werden hierbei vergleichbar mit dem `DataTransfer`-Objekt in HTML5/JavaScript im `DragSource`-Objekt abgelegt, wobei sofort auffällt, dass statt der Daten auch eine Funktion eingetragen werden kann. Diese ermöglicht es, die eigentlichen Daten erst bei Anforderung durch ein potentielles Ziel zusammenzustellen, was bei besonders zeit- oder ressourcenintensiven Operationen sehr sinnvoll erscheint. Eine Übersicht der verschiedenen Ereignisse:

Ereignis	Zuständigkeit	Ausgelöst auf
`dragStart`	Keine, hauptsächlich für listenbasierte Komponenten	Quelle
`dragComplete`	Abschluss etwaiger Verschiebe-Aktionen	Quelle
`dragEnter`	• Akzeptieren oder Abweisen von Drop-Operationen über die Methode `DragManager.acceptDragDrop()` • Visualisierung möglicher Drop-Operationen	Ziel
`dragOver`	• Visualisierung verschiedener Operationen über die Methode `DragManager.showFeedback()`: Kopieren (`DragManager.COPY`), Verschieben (`DragManager.MOVE`), Verlinken (`DragManager.LINK`), keine (`DragManager.NONE`)	Ziel
`dragDrop`	• Auslesen der Daten aus dem `DragSource`-Objekt • Durchführung der angeforderten Operation	Ziel
`dragExit`	Aufhebung der Visualisierung möglicher Drop-Operationen	Ziel

Tabelle 5.2: Drag-and-Drop-Ereignisse in Adobe Flex

Bemerkenswert sind hierbei Komponenten basierend auf der `List`-Komponente[35]. Diese können auf einfachste Art und Weise für Drag-and-Drop-Operationen nutzbar gemacht werden, ohne auch nur ein einziges beteiligtes Ereignis explizit zu verarbeiten. Hier müssen lediglich die beiden Eigenschaften `dragEnabled` und `dropEnabled` auf `true` gesetzt werden. Dadurch registrieren diese Komponenten intern automatisch alle erforderlichen Handler für

[34] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/mx/managers/DragManager.html (abgerufen am 12.06.2011)

[35] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/List.html (abgerufen am 12.06.2011)

Ereignisse und führen die Operationen durch. Soll das Verschieben zusätzlich zum Kopieren möglich sein, kann dies über die `dragMoveEnabled`-Eigenschaft freigeschaltet werden. Für einfache Anwendungen ist der Aufwand hier daher verschwindend gering. Im Flex-Prototypen werden jedoch bewusst die erweiterten Möglichkeiten zur Beeinflussung von Drag-and-Drop-Operationen genutzt. Die List-Klasse wurde wohlweislich konzipiert, um ohne weitreichende Änderungen eigene Datenformate hinzuzufügen. Hierzu ist lediglich ein Überschreiben der Methode `addDragData()` erforderlich:

```
protected function getSelectedResources():Vector.<IResource> {
  var draggedIndices:Vector.<int> = selectedIndices.slice(0, this.selectedIndices.length);
  var resources:Vector.<IResource> = new Vector.<IResource>(draggedIndices.length);
  draggedIndices.sort(compareIndices);
  for (var i:int = 0; i < draggedIndices.length; ++i) {
    resources[i] = this.dataProvider.getItemAt(draggedIndices[i]) as IResource;
  }
  return resources;
}
public override function addDragData(dragSource:DragSource):void {
  dragSource.addHandler(this.getSelectedResources, "resources");
}
```

Listing 5.3: Hinzufügen von Drag-Daten in Adobe Flex

Diese Methode fügt standardmäßig Daten im Formate `itemsByIndex` mit den Datenobjekten der ausgewählten Listeneinträge hinzu. Zusätzlich zur Verfügung steht das Format `caretIndex`, welches auf das Objekt hinweist, mit welchem die Operation initiiert wurde. Im `DragSource`-Objekt steht nun eine getypte Liste von Ressourcen zur Verfügung, diese kann daher in Handlern für `dragEnter`- und `dragOver`-Ereignisse eines Ziels ausgelesen und entsprechende Entscheidungen getroffen werden:

```
protected function onDragOver(e:DragEvent):void {
  e.preventDefault();

  if (e.dragSource.hasFormat("resources")) {
    var resources:Vector.<IResource> =
      e.dragSource.dataForFormat("resources") as Vector.<IResource>;

    if ((MediaRequest.ADD in (this.data as Isource).allowed)
        && resources.every(isFromOtherSource)) {
      this.drawFocus(true);
      DragManager.acceptDragDrop(this);
      DragManager.showFeedback(e.ctrlKey ? DragManager.MOVE : DragManager.COPY);
    } else {
      DragManager.acceptDragDrop(e.dragInitiator);
      DragManager.showFeedback(DragManager.NONE);
    }
  }
}
```

Listing 5.4: dragOver-Handler in Adobe Flex

Hierbei ist anzumerken, dass Kopieren statt Verschieben als Standardverhalten festgelegt

wurde, da dieses nicht destruktiv ist. Über das Betätigen der `[Strg]`-Taste während einer Drag-and-Drop-Operation ist jedoch das Verschieben möglich. Das Auslesen der Daten gestaltet sich damit entsprechend überschaubar:

```
protected function onDrop(e:DragEvent):void {
  this.drawFocus(false);
  if (e.dragSource.hasFormat("resources")) {
    var resources:Vector.<IResource> =
      e.dragSource.dataForFormat("resources") as Vector.<IResource>;
    switch (e.action) {
      case DragManager.COPY:
        var resourcesArray:Array = new Array(resources.length);
        for (var i:int = 0; i < resources.length; ++i) {
          resourcesArray[i] = resources[i];
        }
        MediaManager.instance.addResources(resourcesArray, (this.data as ISource));
        break;
      case DragManager.MOVE:
        MediaManager.instance.moveResources(resources, (this.data as ISource));
        break;
    }
  }
}
```

Listing 5.5: drop-Handler in Adobe Flex

Hierbei wurde jedoch eine Einschränkung deutlich: eine einmal per Aufruf der Methode `DragManager.acceptDragDrop()` für eine Komponente akzeptierte Drop-Operation kann nicht mehr revidiert werden. Während dies im Normalfall kein Problem darstellt, kollidiert es jedoch mit einem Standardverhalten beider Prototypen dieser Arbeit: dem Wechsel zwischen Quellen während einer Drag-and-Drop-Operation. Dieser Wechsel wird beim Überfahren von Einträgen in der Quell-Liste nach einer kurzen Verzögerung vollzogen und erlaubt damit Drop-Operationen direkt in die Ressourcen-Liste von Quellen. Wird nun während einer Drag-and-Drop-Operation durch Überfahren einer Quelle, welche das Hinzufügen von Ressourcen erlaubt, die gesamte Operation akzeptiert, kann diese danach für jede Quelle ausgeführt werden.

Abbildung 5.4: Mögliche Drop-Operation in Adobe Flex

Ein im Kontext dieser Arbeit jedoch weitaus größeres Problem besteht in der Interaktion mit Dateien mit Hilfe von Drag-and-Drop-Operationen. Solcherlei Operationen vom lokalen Speicher in das Flash-Plugin sind nicht möglich und höchstens der AIR-Umgebung vorbehalten [Chambers07]. Dem entsprechend können auch keine Dateien auf diese Art und Weise zu Quellen hinzugefügt werden. Statt dessen muss der in der Flash-Umgebung einzig gangbare Weg eingeschlagen und die Methoden der `FileReference`- und `FileReferenceList`-

Klassen genutzt werden:

> *The FileReference class provides a means to upload and download files between a user's computer and a server. An operating-system dialog box prompts the user to select a file to upload or a location for download.*

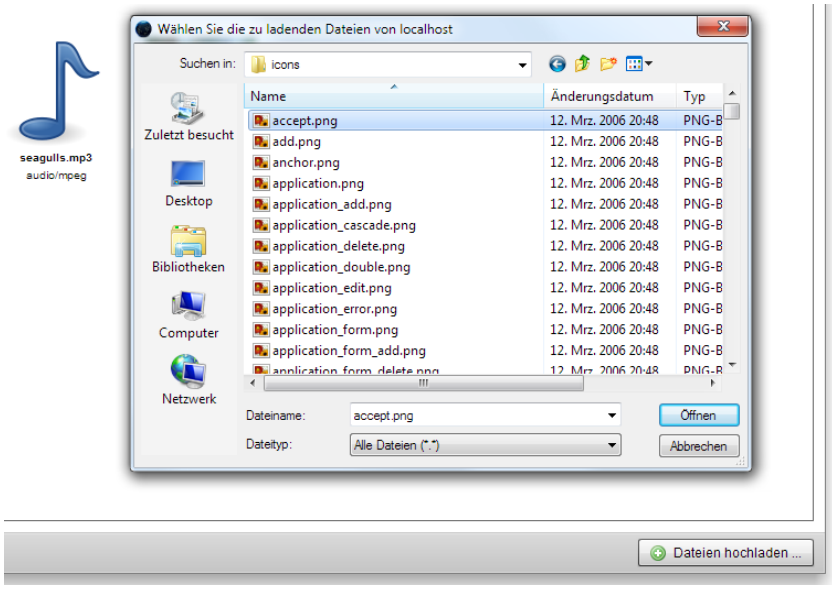

Abbildung 5.5: Dateiauswahldialog in Adobe Flex

Wie aus dieser Kurzbeschreibung der `FileReference`- und `FileReferenceList`-Klasse ersichtlich, übernehmen diese viele Aufgaben auf einmal. Sie sind zuständig für das Beschaffen der Dateien (`FileReference[List].browse()`), das Hochladen von Dateien (`FileReference.upload()`), das Laden von Dateiinhalten (`FileReference.load()`) und sogar das unabhängige Herunterladen von Ressourcen aus dem Web (`FileReference.download()`).

Um das Hochladen von Dateien nichtsdestotrotz zu ermöglichen, wurde eine Schaltfläche hinzugefügt, welche je nach Quelle und erlaubten Aktionen aktiv oder inaktiv ist. Nach Auswahl von Dateien werden diese vor dem Versenden von Anfragen geladen und zur ausgewählten Quelle hinzugefügt.

5.3 Einschätzung

Erstaunlich viele Aspekte der Drag-and-Drop API von HTML5 wurde von der von Microsoft entwickelten und seit Internet Explorer 5 verfügbaren API für Drag-and-Drop [MS-DnD] übernommen, offensichtlich auch aus Gründen der Abwärts-Kompatibilität. Allerdings stellt sich hier die Frage, ob die neue API nicht hätte vereinfacht werden können. Gerade die Vielzahl an Ereignissen stellt eine hohe Herausforderung bzgl. Einarbeitung und Verständnis dar. Dass das Standardverhalten einiger Ereignisse explizit abgebrochen werden muss, um eine Drop-Operation zu ermöglichen, ist auch wenig intuitiv. Da erscheinen Ansätze von Toolkits wie GTK sinnvoller. Hierbei wird im Vorfeld festgelegt, welches Widget als Quelle und

als Ziel agieren kann sowie welche Datentypen und Aktionen möglich sind [GTK-DnD]. Mit Ausnahme von Opera haben eine Vielzahl der großen Browser diese neue API von HTML5 bereits implementiert, auch wenn sie sich noch in der Entwicklung befindet und Änderungen absehbar sind.

Der Zugriff auf Dateien ist eine willkommene Ergänzung und ebnet den Weg für reiche Web-Applikationen, welche die stetig voranschreitende Verschiebung vom Arbeiten auf dem lokalen System in das Web unterstreicht. Beste Beispiele sind hier Google Mail und Google Docs, welche die neuen Möglichkeiten bereits umfangreich zum Einsatz bringen.

In Adobe Flex ist die Tatsache, bei den per Drag-and-Drop übertragbaren Daten nicht auf Zeichenketten beschränkt zu sein, ein hervorzuhebender Vorteil gegenüber der Drag-and-Drop-API in HTML5. Viele Operationen können dadurch direkt durchgeführt werden, anstatt die bspw. durch eine ID referenzierten Daten von anderer Stelle abrufen zu müssen. Auch macht die Aufgabenverteilung der beteiligten Ereignisse einen besser durchdachten Eindruck.

Insbesondere die einfach zuschaltbare Standard-Funktionalität von Listenkomponenten ist ein großes Plus bei der Arbeit mit dem Flex-SDK und erspart in den meisten Fällen viel Zeit. In Bezug auf die Arbeit mit Dateien müssen hier allerdings Abstriche gemacht werden. Dass diese nicht ebenfalls per Drag-and-Drop abgerufen werden können, erfordert viele Umwege und sorgt für eine inkonsistente Bedienung.

6 Ansicht von Ressourcen

Da eine alleinige Verwaltung von Ressourcen wenig sinnvoll ist, ohne diese auch betrachten zu können, ist diese Funktionalität Grundbestandteil der Prototypen. Dabei ist unverzichtbar, dass neben den einfachen Typen Text und Bild auch die Möglichkeit besteht, Audio- und Videoressourcen zu betrachten. Dabei sollten möglichst viele Formate unterstützt werden.

Jedem Ressourcentyp ist ein Betrachter zugeordnet, welcher individuell die gegebenen Möglichkeiten zur Darstellung der betreffenden Ressourcen nutzt. Unter Anwendung des im Rahmen dieser Arbeit entwickelten `ProviderManager` besteht die Möglichkeit, beliebige weitere Betrachter hinzuzufügen oder gar die Standard-Betrachter durch benutzerdefinierte Implementierungen zu ersetzen. Unerwünschte Betrachter können hierdurch aber auch entfernt werden. Beide Implementierungen machen hiervon Gebrauch.

6.1 Textdaten und Bilder

Die einfachsten aller Ressourcen-Typen gehören zur Standardausrüstung jedes guten Betrachters und sind üblicherweise ohne großen Aufwand integrierbar.

6.1.1 HTML5/JavaScript

Um die Anforderungen an das Scripting niedrig zu halten, kommt für die Anzeige von Textinhalten ein eingebetter Frame zum Einsatz, die Auslieferung der korrekten HTTP-Header obliegt hierbei dem Server der Quelle:

```
function PlainTextResourceFormatter() {
  this.format = function(resource) {
    var $item = $("<iframe/>");
    $item.attr({ src: resource.fullURL, seamless: "seamless", height: 600, width: 400 });
    return $item;
  };
}
```

Listing 6.1: Formatierer für Text-Ressourcen in HTML5/JavaScript

Der Betrachter für Bild-Ressourcen gestaltet sich, bedingt durch die Tatsache, dass Bilder schon seit jeher zum Repertoire der im Web nutzbaren Ressourcen gehören, vergleichbar einfach:

```
function ImageResourceFormatter() {
  this.format = function(resource) {
    var $item = ImageResourceFormatter.prototype.format.call(this, resource, resource.fullURL);
    $item.attr("alt", resource.properties.title).data("enlarge", true);
    return $item;
  };
}
ImageResourceFormatter.prototype = new ImageFormatter;
```

Listing 6.2: Formatierer für Bild-Ressourcen in HTML5/JavaScript

Die zugehörige Basisklasse übernimmt hierbei den Hauptteil der Arbeit und zeigt während des Ladeversuchs des Bilds eine Ladegrafik an. Falls ein Bild nicht geladen werden kann, wird mit Hilfe des MIME-Typs der Ressource versucht, ein Symbol als Platzhalter zu finden. Schlägt selbst dies fehl, komm ein generisches Symbol zum Einsatz:

```
function ImageFormatter() {
  var types = {"application/msword": "document.png", /* ... */ "audio": "audio.png", /* ... */};
  function guessFallback(type) {
    var fallback = "images/thumbnails/generic.png";
    if (typeof type !== "undefined") {
      var type = type.split("/")[0];
      if (type in types) {
        fallback = "images/thumbnails/" + types[type];
      }
    }
    return fallback;
  }
  this.format = function(resource, value) {
    var $image = $("<img/>");
    var type = resource.properties.type;
    if (!value) { var value = guessFallback(type); }
    else if (!/^[a-z]+:\/\//.test(value)) { value = resource.source.url + "/" + value; }

    $image.addClass("loading")
          .bind("load", function() { $(this).removeClass("loading"); })
          .bind("error", {type: type}, function(e) {
            $(this).attr("src", guessFallback(e.data.type));
          })
          .attr("src", value).attr("draggable", false);
    return $image;
  };
}
```

Listing 6.3: Allgemeiner Formatierer für Bilder in HTML5/JavaScript

Da das `img`-Element die beiden Ereignisse `load` bei Erfolg und `error` im Fehlerfall auslöst, lassen sich hierdurch sämtliche Fälle abdecken.

6.1.2 Adobe Flex

Zum Betrachten von textlichen Inhalten wird die `URLLoader`-Klasse verwendet, wobei der Betrachter selbst von der `Textarea`-Komponente abgeleitet ist. Nach dem Laden der Text-Ressource wird nur noch die `text`-Eigenschaft der Komponente gefüllt.

Die Spark `Image`-Komponente[36] erlaubt es durch die interne Verwendung der `BitmapImage`-Klasse Bilder von nahezu beliebigen Quellen einzubinden. Hierbei inbegriffen sind Bilder, welche über Web-Adressen abzurufen sind. Über die Eigenschaft `enableLoadingState` lässt sich ein vorgefertigter Ladebalken einblenden und über `smooth` und `smoothingQuality` die automatische Glättung der geladenen Grafiken aktivieren.

[36] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/Image.html (abgerufen am 12.06.2011)

6.2 Audiodaten

Für die Betrachtung von Audio-Ressourcen ist es erforderlich, eine grafische Komponente für die Kontrolle der Wiedergabe einzusetzen. Diese muss sowohl die Wiedergabe starten und pausieren sowie die Lautstärke regeln können als auch das gezielte Springen an eine beliebige Zeitposition innerhalb der Audio-Ressource erlauben. Eine Visualisierung des Audio-Spektrums (Equalizer) gehört mittlerweile zum Standard und ist daher ebenso empfehlenswert.

6.2.1 HTML5/JavaScript

Eine der bemerkenswertesten Neuerungen in HTML5 ist die native Unterstützung der Wiedergabe von Audio-Ressourcen. In Form des von Microsoft proprietär entwickelten `bgsound`-Elements gab es bereits einmal eine Möglichkeit, Audio-Ressourcen ohne Plugins wiederzugeben. Die Möglichkeiten der Interaktion mit diesem Element waren allerdings praktisch nicht vorhanden. Das HTML5 `audio`-Element[37] dagegen bietet über die Schnittstelle `HTMLMediaElement` eine Vielzahl an Funktionen zur Steuerung der Wiedergabe. Dies umfasst neben Wiedergabe und Pause, der Anpassung der Lautstärke inkl. Stummschaltung und der Möglichkeit des Anspringens beliebiger Zeitpunkte die Festlegung, ob die Ressource automatisch sowie nach Wiedergabe automatisch wieder von vorn abgespielt und in welcher Form vor der Wiedergabe gepuffert werden soll.

Zudem ist die wiederzugebende Audio-Ressource nicht auf eine beschränkt, vielmehr können verschiedene Formate als Alternativen angegeben werden. Zusätzlich besteht die Möglichkeit, verschiedene Audio-, Video- und Text-Tracks parallel wiederzugeben. Die Angabe verschiedener Versionen der Audio-Ressource ist leider unverzichtbar, da alle Browser unterschiedliche Audio-Formate und -Codecs unterstützen. Mit Hilfe des `source`-Elements werden verschiedene Versionen angegeben und können um die Angabe des MIME-Typs und der erforderlichen Codecs ergänzt werden. Eine Übersicht über verfügbare Audio-Formate und -Codecs und ihre Unterstützung in den aktuellen Browsern ist ausführlich im Wiki der WHATWG zu finden [VideoParameters]. Das WebM-Format ist hierbei das neueste und wurde von Google mit dem Ziel eines gemeinsamen Standard-Formats im Sinn entwickelt und umfasst einen auf dem Matroska-Format basierten Container, den Audiocodec Vorbis und den Videocodec VP8 [WebM]. Der Grund, weshalb sich die Browserhersteller nicht auf ein Format geeinigt haben, liegt zum einen in den konkreten Implementierungen und zum anderen bei lizenzrechtlichen Einschränkungen hinsichtlich Formaten und Codecs.

Standardmäßig verfügt das `audio`-Element über keine Bedienelemente und kann nur per JavaScript-API gesteuert werden. Beim Setzen des `controls`-Attributs, zeigen dagegen alle aktuellen Browser eigene Bedienelemente an:

[37] http://www.w3.org/TR/2011/WD-html5-20110525/the-iframe-element.html#the-audio-element (abgerufen am 18.06.2011)

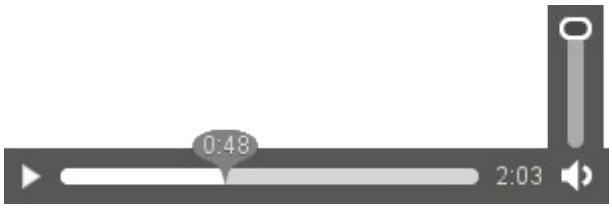

Abbildung 6.1: HTML5-Audio-Bedienelemente im Mozilla Firefox

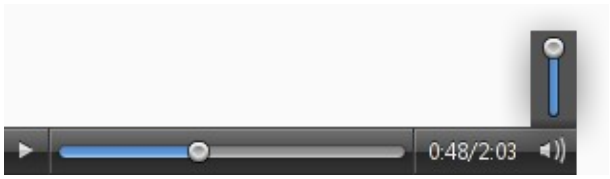

Abbildung 6.2: HTML5-Audio-Bedienelemente im Opera

Abbildung 6.3: HTML5-Audio-Bedienelemente in Google Chrome

Abbildung 6.4: HTML5-Audio-Bedienelemente im Internet Explorer

Abbildung 6.5: HTML5-Audio-Bedienelemente im Apple Safari

Im Rahmen der Entwicklung des Prototypen wurde für die Wiedergabe von Audio-Ressourcen allerdings auf diese Standard-Bedienelemente verzichtet und stattdessen eigene entwickelt:

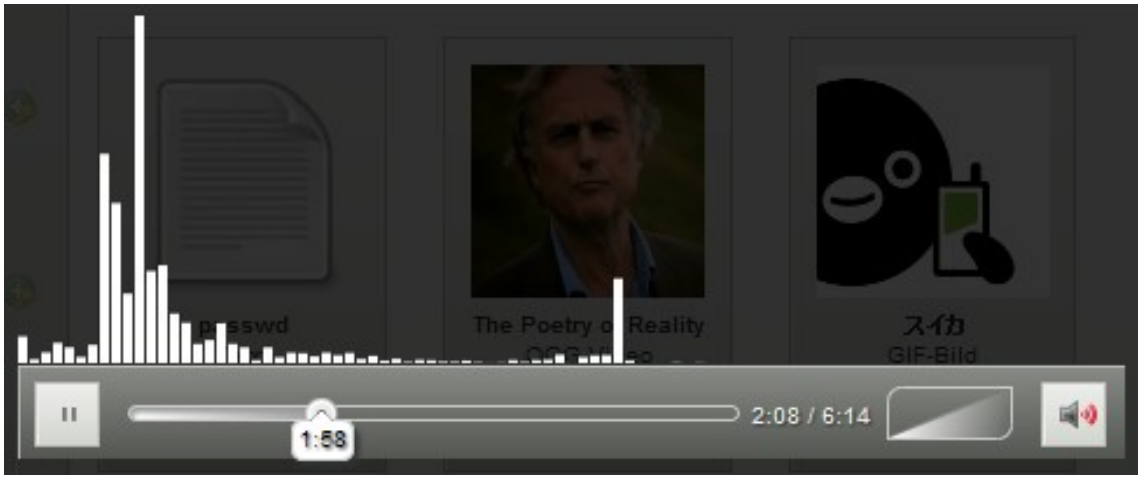

Abbildung 6.6: Audio-Wiedergabe in HTML5/JavaScript

Bei den Schaltflächen für Wiedergabe/Pause und Stummschaltung handelt es sich um reguläre `button`-Elemente. Für die Lautstärkeregelung kommt das `canvas`-Element in einem `VolumeWidget` zum Einsatz, um den Pegel deutlich zu visualisieren. Die Suchleiste wird durch das `ProgressSeekWidget` bereitgestellt, wobei der gefüllte Bereich durch ein Element repräsentiert wird, dessen Breite über das `timeupdate`-Ereignis an den Wiedergabefortschritt des `audio`-Elements angepasst wird. Dadurch wandert implizit der Anfasser mit. Dessen Infotext wird während des Suchens angezeigt und informiert über die Position, die nach dem

Loslassen angesprungen wird.

Angelehnt an übliche Audio-Wiedergabeprogramme erscheint es nur logisch, dass die wiedergegebenen Audiodaten visualisiert werden. Die grafische Darstellung der einzelnen Teilfrequenzen in Form eines Spektrums ist hierfür am besten geeignet. Bedauerlicherweise bietet das `audio`-Element in HTML5 momentan keine Möglichkeit, auf den internen Wiedergabepuffer und Eckdaten wie die Anzahl der Kanäle und die verwendete Abtastrate zuzugreifen.

In Form der Audio Data API von Mozilla gibt es jedoch eine Erweiterung, die genau diese Einschränkung aufhebt [AudioDataAPI]. Diese Erweiterung erlaubt neben dem Auslesen des Wiedergabepuffers auch das Beschreiben und damit bspw. das Anwenden von Effekten sowie die komplette programmgesteuerte Erzeugung von Ton. Da diese API momentan noch eine Eigenentwicklung von Mozilla ist, sind die vorgeschlagenen Eigenschaften und Ereignisse mit dem Präfix „`moz`“ versehen. Der HTML5/JavaScript-Prototyp prüft auf das Vorhandensein dieser Eigenschaften und nutzt diese, um die Visualisierung vorzubereiten:

```
$audio.bind({
  /* ... */
  loadedmetadata: function() {
    if (!"mozFrameBufferLength" in player) return;
    $equalizer.slideDown();
    var frameBufferLength = player.mozFrameBufferLength;
    var channels = player.mozChannels;
    var fft = new FFT(frameBufferLength / channels, player.mozSampleRate);
    var context = $equalizer[0].getContext("2d");
    context.fillStyle = "#ffffff";
    // Process signals and visualize them
    $audio.bind("MozAudioAvailable", function(e) {
      var e = e.originalEvent;
      var signal = new window[(("Float32Array" in window) ? "Float32Array" : "Array")]
(frameBufferLength / channels); // Prefer Float32Array if available
      for (var i = 0; i < frameBufferLength / channels; ++i) {
        signal[i] = (e.frameBuffer[channels * i] + e.frameBuffer[channels * i + 1]) / channels;
      }
      fft.forward(signal); // Do FFT
      var equalizerHeight = $equalizer[0].height;
      context.clearRect(0, 0, $equalizer.width(), $equalizer.width());
      for (var i = 0; i < fft.spectrum.length; ++i) {
        var normalizedValue = Math.max(0, Math.min((fft.spectrum[i] * 10), 1));
        var rectangle = new Rectangle(
          i * 5, equalizerHeight,
          4, equalizerHeight * normalizedValue
        );
        context.fillRect(rectangle.left, rectangle.top, rectangle.width, -rectangle.height);
      }
    });
  }
});
```

Listing 6.4: Visualisierung von Audio-Daten in HTML5/JavaScript

Die Kanäle des rohen Audio-Signale werden zunächst über den Durchschnitt zusammengelegt und danach mittels Fast Fourier Transformation in einen geeigneten Wertebereich umgewandelt. Hierfür kommt die JavaScript-Bibliothek dsp.js zum Einsatz, welche eine Fülle an Funktionen für die Analyse und Erzeugung von Signalen enthält [DSP-JS]. Auf dem `canvas`-Kontext beeinflussen diese Werte schließlich die Position und Höhe der gezeichneten Balken, von niedrigen Frequenzen im linken bis hohen Frequenzen im rechten Bereich.

6.2.2 Adobe Flex

Basierend auf der `Sound`-Klasse[38] stehen im Flex-SDK verschiedene Möglichkeiten zur Verfügung, Audio-Ressourcen wiederzugeben. Als Teil des Flex-SDK ist dabei das Open Source Media Framework [OSMF] eine quelloffene Entwicklung von Adobe, die darauf abzielt, die Entwicklung von Komponenten zur Wiedergabe von Medien zu vereinfachen. Standardmäßig ist das OSMF bei der Unterstützung von Audio-Formaten und -Codecs auf die Möglichkeiten des Flash-Players[39] angewiesen. Im Rahmen des Flex-Prototypen wird die vom OSMF zur Verfügung gestellte `MediaPlayer`-Klasse[40] genutzt.

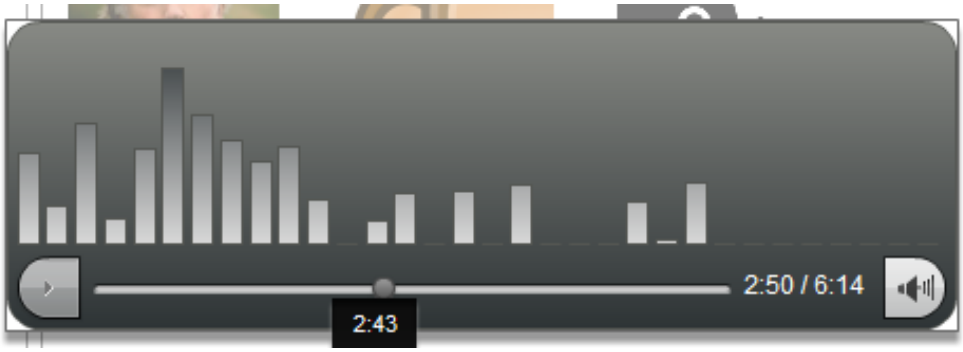

Abbildung 6.7: Audio-Wiedergabe in Adobe Flex

Bedauerlicherweise wurde das OSMF nicht soweit in das Flex-SDK integriert wie es zu erwarten wäre: keine der in den Framework-Klassen definierten Eigenschaften wurde für Databindings freigegeben. Ein automatisches Aktualisieren von Komponenten basierend auf dem Zustand der OSMF-Klassen ist daher nicht möglich. Die althergebrachte Verarbeitung aller zur Verfügung stehenden Ereignisse ist daher unumgänglich, wie an der Deklaration des `MediaPlayer`-Objekts deutlich wird:

```
<osmf:MediaPlayer
  id="player" autoPlay="false" autoRewind="true"
  hasAudioChange="onHasAudioChange(event)"
  mediaPlayerStateChange="onPlayerStateChange(event)"
  currentTimeChange="onCurrentTimeChange(event)"
  durationChange="onDurationChange(event)"/>
```

Listing 6.5: Deklaration des MediaPlayer in Adobe Flex

[38] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/media/Sound.html (abgerufen am 12.06.2011)
[39] http://kb2.adobe.com/cps/402/kb402866.html (abgerufen am 12.06.2011)
[40] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/org/osmf/media/MediaPlayer.html (abgerufen am 12.06.2011)

Die Zuständigkeiten der einzelnen Handler sind dennoch überschaubar. So erlaubt die Verarbeitung des `hasAudioChange`-Ereignisses die Anpassung der Lautstärke-Komponente, das `mediaPlayerStateChange`-Ereignis betrifft die Wiedergabe/Pause-Schaltfläche und das `durationChange`-Ereignis bestimmt den Maximalwert der Suchleiste. Beim Auslösen des `currentTimeChange`-Ereignisses wird allerdings neben dem aktuellen Wert der Suchleiste und dem Text zur Anzeige der Wiedergabeposition auch der Equalizer angewiesen, seinen Inhalt neu zu zeichnen.

Der Equalizer ist für die Visualisierung der wiedergegebenen Audio-Daten zuständig. Hierzu überschreibt diese Komponente die Methode `updateDisplayList()` der `UIComponent`-Klasse für eigene Zeichenoperationen. Für die Ermittlung des aktuellen Frequenzspektrums steht die `SoundMixer`-Klasse[41] mit der Methode `computeSpectrum()` zur Verfügung. Diese kann auf die Audio-Daten vor der Ausgabe auch automatisch eine FFT anwenden. Beachtet werden muss hier, dass die `SoundMixer`-Klasse sich auf die Tonausgabe der gesamten Applikation bezieht, ein gezieltes Verarbeiten einzelner Signale ist daher nicht möglich.

```
protected override function updateDisplayList(unscaledWidth:Number, unscaledHeight:Number):void {
  var soundBuffer:ByteArray = new ByteArray();
  SoundMixer.computeSpectrum(soundBuffer, true);
  this.graphics.clear();

  for (var i:uint = 0, bufferLength:uint = (soundBuffer.length / 8); i < bufferLength; i += 8) {
    var value:Number = soundBuffer.readFloat();
    var gradientMatrix:Matrix = new Matrix();
    // Create matrix for a 10*height bar, rotated by 90 degrees
    gradientMatrix.createGradientBox(10, height, 90);
    // Set stroke and fill colors
    this.graphics.lineStyle(1, 0x555753);
    this.graphics.beginGradientFill(
      GradientType.LINEAR,
      [0x2e3436, 0xd9d9d9],
      [1, 1],
      [0, 255],
      gradientMatrix
    );
    // Draw the bar based on the current value
    this.graphics.drawRect(
      width * (i / bufferLength), // E.g. 300 * (8 / 256) = 9
      height, // E.g. 100
      10,
      -(height * value) // E.g. -(100 * 0.04504564)
    );
    this.graphics.endFill();
  }
}
```

Listing 6.6: Visualisierung von Audio-Daten in Adobe Flex

Zum Zeichnen der einzelnen Werte wird der ausgelesene Wiedergabepuffer durchlaufen und jeder Wert als Fließkommazahl ausgelesen. Dieser Wert im Bereich [0..1] bestimmt schließlich

[41] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/media/SoundMixer.html (abgerufen am 12.06.2011)

die Höhe der gezeichneten Balken. Diese werden unter Verwendung der `Graphics`-Klasse[42] und deren Methoden zum Generieren vektorbasierter Grafiken gezeichnet. Hier kommen ein einfacher grauer Farbverlauf und eine umrahmende Linie zum Einsatz.

6.3 Videodaten

Vergleichbar mit der Wiedergabe von Audio-Ressourcen bedarf es einer grafischen Komponente zur Beeinflussung der Video-Wiedergabe. Diese ist zuständig für die synchrone Wiedergabe von Bild und Ton.

6.3.1 HTML5/JavaScript

Bislang ausschließlich mit Hilfe von Plugins möglich, ist mit der HTML5-Spezifikation nun auch die native Wiedergabe von Video-Ressourcen in das Web eingezogen. Das `video`-Element[43] verfügt über die Schnittstelle `HTMLMediaElement` über die gleichen Möglichkeiten zur Kontrolle der Wiedergabe wie das `audio`-Element. Die gleichen Anmerkungen bzgl. verschiedener Quellen sowie Audio-Formate und -Codecs gelten daher auch hier. (Siehe Abschnitt 6.2.1)

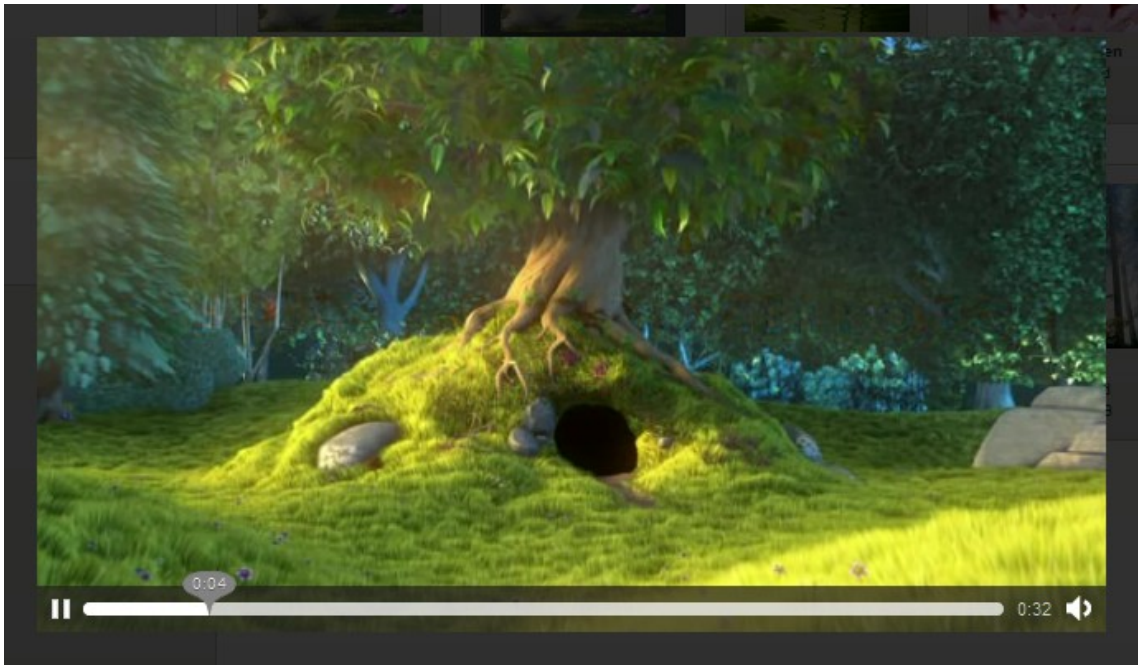

Abbildung 6.8: Video-Wiedergabe in HTML5/JavaScript

Zusätzlich bietet dieses Element Attribute für die Festlegung der Höhe und Breite, Zugriff auf die Original-Ausmaße eines Videos und die Angabe eines Platzhalter-Bildes (`poster`). Dieses wird angezeigt, so lange noch keine Videodaten geladen wurden. Für die Anzeige von Video-Ressourcen werden im HTML5/JavaScript-Prototyp die Standard-Bedienelemente der Browser genutzt:

[42] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/display/Graphics.html (abgerufen am 12.06.2011)

[43] http://www.w3.org/TR/2011/WD-html5-20110525/the-iframe-element.html#the-video-element (abgerufen am 18.06.2011)

```
function VideoResourceFormatter() {
  this.format = function(resource) {
    var $item = $("<video/>");
    $item.bind({ removed: function() { $item[0].src = null; } });
    $item.attr({ src: resource.fullURL, autoplay: "autoplay", controls: "controls" })
         .html("<code>video</code>-Element nicht verfügbar.")
         .data("enlarge", true);
    return $item;
  };
}
```

Listing 6.7: Formatierer für Video-Ressourcen in HTML5/JavaScript

Wie bereits bei Audio-Ressourcen setzt auch der Formatierer für Video-Ressourcen beim Empfang des von der `Viewer`-Klasse initiierten `removed`-Ereignisses die `src`-Eigenschaft auf `null`. Damit ist sichergestellt, dass die Wiedergabe nicht nach dem Ausblenden des Betrachters weiter läuft.

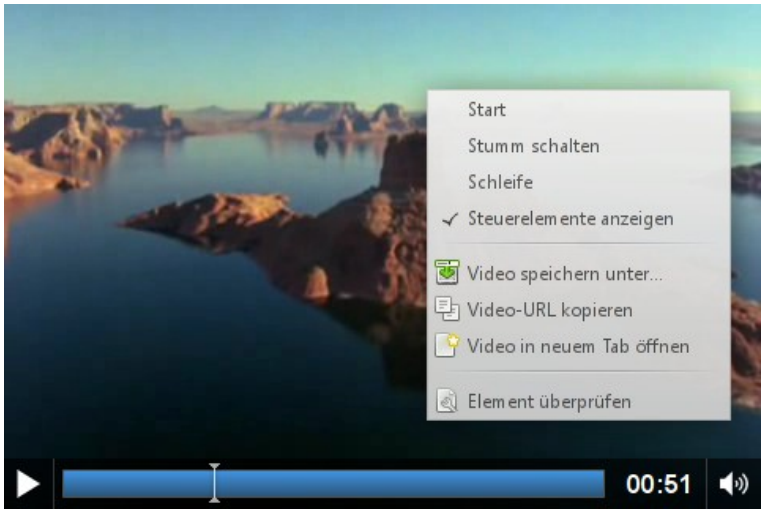

Abbildung 6.9: Kontextmenü von Video-Elementen in HTML5/JavaScript

Alle Browser bieten von Haus aus über das Kontextmenü der Standard-Bedienelemente die Möglichkeit, die Wiedergabe zu steuern. Zusätzlich ermöglichen alle Browser auf diese Weise, die Adresse der abgespielten Audio- oder Video-Ressource zu kopieren oder diese herunterzuladen und auf dem lokalen Speicher abzulegen.

6.3.2 Adobe Flex

Für die Video-Wiedergabe Flex-Implementierung des Prototypen wird die Spark `VideoPlayer`-Komponente[44] genutzt. Diese definiert sich wie folgt:

> *The VideoPlayer control is a skinnable video player that supports progressive download, multi-bitrate streaming, and streaming video. It supports playback of FLV and F4v files. The VideoPlayer control contains a full-featured UI for controlling video playback.*

[44] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/VideoPlayer.html (abgerufen am 12.06.2011)

Abbildung 6.10: Video-Wiedergabe in Adobe Flex

Erwähnenswert in Bezug auf die `VideoPlayer`-Komponente ist der Zugriff auf das interne `Video`-Objekt. Hierüber kann unter anderem das mit dem Flash Player 9.0.115.0 eingeführte Mip-Mapping aktiviert werden, was die Bildqualität bei skalierten Videos durch Interpolation und Glättung merklich verbessert. Die standardmäßig in der `VideoPlayer`-Komponente verfügbare Schaltfläche zum Wechsel in den Vollbildmodus wurde mit Hilfe eines benutzerdefinierten Skins entfernt, da der Ressourcenbetrachter diese Funktionalität selbst implementiert.

6.4 Einschätzung

Die Betrachtung der einfachen Typen Text und Bild bereitet keiner der beiden Implementierungen Probleme. Interessanter ist die veränderte Situation in Bezug auf die Wiedergabe von Audio- und Videoressourcen durch die mit der HTML5-Spezifikation eingeführten Elemente. Während diese Funktionalität früher einzig und allein Plugins und hier insbesondere dem Flash-Player vorbehalten war, besteht nun die Möglichkeit der nativen Wiedergabe direkt im Browser ohne den Bedarf an zusätzlichen Plugins. Der reichhaltigen Möglichkeiten in Adobe Flex, die Erscheinung der Wiedergabekomponente durch Stile und Skins anzupassen, steht HTML5 in nichts nach. Mittels JavaScript und CSS können einfache bis komplexe Oberflächen erstellt und bei Bedarf mittels des `canvas`-Elements um handgezeichnete Teile ergänzt werden.

Beim konkreten Vergleich der nativen Wiedergabe von Ressourcen in Browsern und der Wiedergabe mit Hilfe des OSMF in Adobe Flex fällt jedoch ein Schwachpunkt bei der HTML5-Implementierung ins Auge: welche Ressourcen unterstützt und damit wiedergegeben werden, hängt stark von den Implementierungen der Browser ab. Unterstützen diese ein bestimmtes

Format oder einen Codec nicht, kann keine Abhilfe geschaffen werden. Manche Browser erlauben die Erweiterung der Fähigkeiten; Apples Safari-Browser spielt bspw. alles ab, was vom hauseigenen Quicktime-Player unterstützt wird. Ohne installierten Quicktime-Player kann Safari dem entsprechend keinerlei Medien wiedergeben. Andere Browser nutzen bestehende Media-Frameworks; Googles Chrome-Browser nutzt die FFmpeg-Bibliothek [FFmpeg], Opera das GStreamer-Framework [Opera-GStreamer]. Beide unterstützen damit theoretisch mehr Formate als für das Web relevant sind. Für alle anderen Fälle gibt es jedoch keine Möglichkeit, eine Unterstützung für unbekannte Formate und Codecs hinzuzufügen. Beim OSMF in Flex gestaltet sich dies anders. Dieses Framework unterstützt explizit die Erweiterung über Plugins[45], auf diesem Wege kann also auch die Unterstützung für unbekannte Formate hinzugefügt werden.

Ungeachtet dieses Aspekts hat HTML in seiner fünften Version durch neue Elemente und APIs aufgeholt und erlaubt vielerorts den Verzicht auf Plugins wie Flash durch den Einsatz nativer Lösungen. Zu sehen unter anderem am Beispiel Youtube, der wohl populärsten Video-Plattform im Web. Diese bietet seit geraumer Zeit die Möglichkeit, HTML5 für die Videowiedergabe zu bevorzugen [Youtube-HTML5].

[45] http://sourceforge.net/adobe/osmf/wiki/Plugins/ (abgerufen am 12.06.2011)

7 Sitzungsübergreifende Persistierung von Daten

Schon seit Anbeginn des Web besteht der Bedarf, benutzerdefinierte Informationen von Nutzern über mehrere Sitzungen hinweg zu erhalten. Beispiele hierfür sind gespeicherte Artikel im Warenkorb eines Online-Shops, Informationen über den Login-Zustand in Foren und dergleichen.

7.1 HTML5/JavaScript

Im Kontext Web ist der bisher am weitesten verbreitete Weg, Daten sitzungsübergreifend zu sichern, die Nutzung von HTTP-Cookies. Diese stellen kleine Speichereinheiten zur Speicherung von textlichen Informationen dar und wurden ursprünglich von der Firma Netscape entwickelt. Aktuell spezifiziert sind Cookies und ihre Eigenschaften in Form der RFC 6265 [RFC6265]. Alternative Lösungen bestehen in der serverseitigen Zuweisung einer Sitzungs-ID an einen Client. Dieser Sitzungs-ID werden die betreffenden Daten zugeordnet. Verschiedene Methoden können zum Einsatz kommen, um die Assoziation eines Clients zu einer Sitzungs-ID aufrecht zu erhalten. Neben dem Anhängen der ID an sämtliche Verweise in einem HTML-Dokument stellen auch HTTP-Cookies eine solche Methode dar.

Während es serverseitigen Technologien durchaus möglich ist, für eine Domain mehrere Cookies zu setzen, beschränkt sich dies bei JavaScript auf eines. Es müssen Umwege beschritten werden, um multiple Werte in dieses eine Cookie zu schreiben und jede Stelle, die auf das Cookie zugreift, muss dies konsistent berücksichtigen. Zudem ist problematisch, dass beim Aufruf einer Ressource innerhalb einer Domain immer bedingungslos sämtliche Cookies an den Server gesendet werden. Dies erzeugt unnötige Datenübertragungen, welche zudem ungesichert erfolgen. Bei der generellen Kommunikation mittels SSL ist dieses Manko natürlich nicht relevant. Die übertragene Datenmenge kann sich durchaus bemerkbar machen, da Cookies gemäß Spezifikation eine Größe von mindestens 4KB erlauben.

Die Fülle an durch HTML5 eingeführten neuen APIs für Datenspeicherung und -verwaltung innerhalb von Browsern bietet Lösungen für all diese Beschränkungen. Hierzu zählt neben der Web Storage API [WebStorage] auch die Indexed Database API [IndexedDB]. Die Arbeit an der vergleichbaren Web SQL Database API[46] wurde zugunsten der Indexed Database API eingestellt. Während die Indexed Database API das Speichern größerer und strukturierter Daten erlaubt, beschränkt sich die Web Storage API auf einfache Operationen über Schlüssel-Wert-Paare.

Da sich die zu persistierenden Daten im Rahmen dieser Arbeit auf Zustände zum Wiederherstellen bei der nächsten Browsersitzung beschränken, wird die Web Storage API zur Speicherung einfacher Daten eingesetzt. Den Unterschied der darin definierten Objekte

[46] http://www.w3.org/TR/2010/NOTE-webdatabase-20101118/

`sessionStorage` und `localStorage` beschreibt die Spezifikation wie folgt:

> *The [`sessionStorage`] is designed for scenarios where the user is carrying out a single transaction, but could be carrying out multiple transactions in different windows at the same time. [...]*
>
> *The [`localStorage`] mechanism is designed for storage that spans multiple windows, and lasts beyond the current session. In particular, Web applications may wish to store megabytes of user data, such as entire user-authored documents or a user's mailbox, on the client side for performance reasons.*
>
> *Each site has its own separate storage area.*

Beide Objekte implementieren die `Storage`-Schnittstelle und damit Methoden zum Setzen, Lesen und Löschen von Schlüssel-Wert-Paaren. Der Einfachheit halber sind die gleichen Operationen auch mit Standard-Operatoren von JavaScript möglich. Folgende Beispiele sind daher äquivalent:

```
// null
alert(localStorage.getItem("myKey"));
localStorage.setItem("myKey", "myValue");
// "myValue"
alert(localStorage.getItem("myKey"))
localStorage.removeItem("myKey");
// null
alert(localStorage.getItem("myKey"));
```

Listing 7.1: Methodenbasierter Zugriff auf localStorage in HTML5/JavaScript

```
// null
alert(localStorage["myKey"]);
localStorage["myKey"] = "myValue";
// "myValue"
alert(localStorage["myKey"]);
delete localStorage["myKey"];
// null or undefined
alert(localStorage["myKey"]);
```

Listing 7.2: Operatorbasierter Zugriff auf localStorage in HTML5/JavaScript

Problematisch ist hierbei, dass das `localStorage`-Objekt für alle Scripte in einem Dokument verfügbar ist. Ein fremdes Script kann daher auf einfache Weise über das `localStorage`-Objekt iterieren und sämtliche darin abgelegten Informationen auslesen. Dass hier keine sensitiven Informationen abgespeichert werden sollten, dürfte aus diesem Umstand ersichtlich werden.

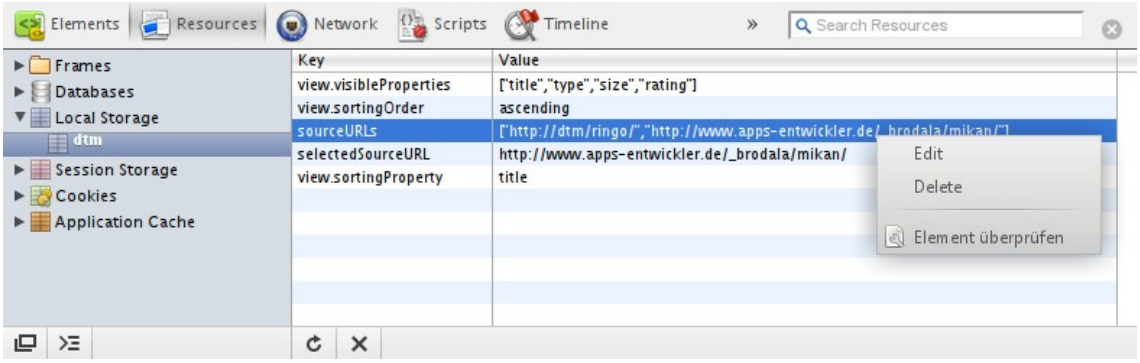

Abbildung 7.1: Speicherverwaltung in Google Chrome

Das `localStorage`-Objekt wird vom `SettingsManager` für die Persistierung von Daten verwendet. Bedingt durch die Beschränkung auf Zeichenketten ist es erforderlich, Objekte zu

serialisieren. Hierfür sind verschiedene Filter integriert, welche zu speichernde Werte in das erforderliche Format umwandeln. Dies umfasst einen Filter für die Umwandlung von Objekte in JSON und einen Filter, welcher das einfache Speichern der Liste hinzugefügter Quellen erlaubt. Neben dieser Liste wird auch die zuletzt gewählte Quelle und in der Ressourcen-Liste die Sortierreihenfolge sowie die anzuzeigenden Eigenschaften von Ressourcen gespeichert. Beim nächsten Aufruf des Prototypen werden diese Einstellungen ausgelesen und damit der letzte Zustand wiederhergestellt.

Die Unterstützung für das `localStorage`-Objekt in den verschiedenen Browsern ist vorbildlich:

	 **Mozilla Firefox**	 **Microsoft Internet Explorer**	 **Google Chrome**	 **Opera**	 **Apple Safari**
Unterstützung ab	3.5[47]	8.0[48]	4.0[49]	10.5[50]	4.0[51]
Max. Speicherplatz	5MB	10MB	5MB	5MB	5MB

Tabelle 7.1: Unterstützung des HTML5 localStorage-Objekts in Browsern

7.2 Adobe Flex

Um in der Flash-Umgebung Daten über mehrere Sitzungen hinweg zu persistieren, werden seit Version 6 sog. Local Shared Objects (LSO), im Allgemeinen auch „Flash-Cookies“ verwendet [SharedObject]. In ActionScript verfügbar werden diese über die `SharedObject`-Klasse[52], welche sich wie folgt definiert:

> *The SharedObject class is used to read and store limited amounts of data on a user's computer or on a server. Shared objects offer real-time data sharing between multiple client SWF files and objects that are persistent on the local computer or remote server.*

Standardmäßig stehen einer Domain 100KB Speicherplatz zur Verfügung, welcher jedoch mit der Zustimmung des Nutzers erhöht werden kann. Der Teilbegriff „Shared“ verdeutlicht hierbei den markantesten Unterschied zu den regulären HTTP-Cookies: verschiedene Clients teilen sich das gleiche Objekt. Dadurch ist es bspw. problemlos möglich, in einem Browser Aktionen vorzunehmen, welche das Speichern von Informationen in einem LSO bewirkt. Bei Aufruf der selben Applikation in einem anderen Browser steht sodann die zuvor gespeicherte Information zur Verfügung. Dies geschieht hierbei in Echtzeit.

Dieser Effekt gründet darauf, dass der Flash-Player Daten über die LSO in einem eigenen

47 https://developer.mozilla.org/en/dom/storage#localStorage (abgerufen am 13.06.2011)

48 http://msdn.microsoft.com/en-us/library/cc197062(VS.85).aspx (abgerufen am 13.06.2011)

49 http://googlechromereleases.blogspot.com/2010/01/stable-channel-update_25.html (abgerufen am 13.06.2011)

50 http://dev.opera.com/articles/view/web-storage/ (abgerufen am 13.06.2011)

51 http://developer.apple.com/library/safari/#documentation/iPhone/Conceptual/SafariJSDatabaseGuide/Name-ValueStorage/Name-ValueStorage.html (abgerufen am 13.06.2011)

52 http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/SharedObject.html (abgerufen am 13.06.2011)

Speicherbereich unabhängig von Browsern verwaltet. Dieser Fakt hat den LSO allerdings auch ein negatives Image verschafft, da normalen Nutzern nicht ohne Weiteres klar ist, wie man diese Informationen löschen kann. Dies ist nur durch den Aufruf des Einstellungsmanagers von Adobe[53] oder durch manuelles Löschen der betreffenden Dateien in den lokalen Verzeichnissen möglich. Programme von Drittanbietern vereinfachen diesen Prozess.

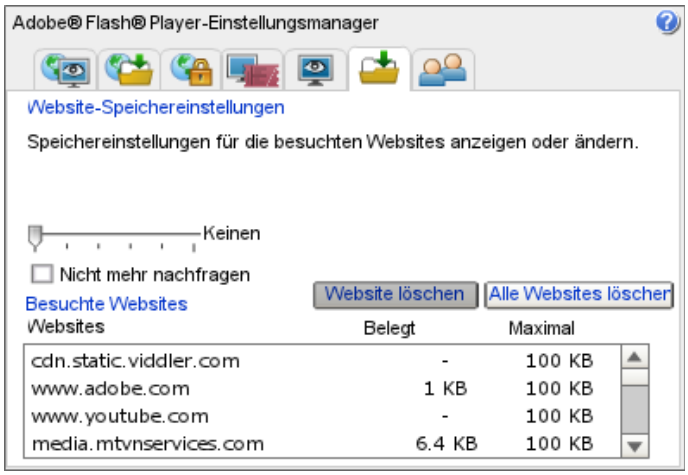

Abbildung 7.2: Einstellungsmanager des Adobe Flash Player

Der `SettingsManager` nutzt diese LSO und macht sich die Tatsache zunutze, dass hiermit nicht nur einfache Zeichenketten sondern beliebige Objekte gespeichert werden können. Folgendes ist daher problemlos möglich:

```
var storage:SharedObject = SharedObject.getLocal("mydata");
storage.data.simpleData = 42;
storage.data.arrayData = ["one", "two", "three"];
storage.data.customData = new CustomObject();
```

Listing 7.3: Speicherung beliebiger Daten durch LSO in Adobe Flex

Über die `data`-Eigenschaft werden neue Einträge hinzugefügt und mit Hilfe des `delete`-Schlüsselworts gelöscht. Beachtet werden muss hier allerdings, dass Objekte beim Speichern in LSO mit Hilfe des Action Message Formats (AMF) serialisiert werden. Dies hat zur Folge, dass beim späteren Auslesen statt der ursprünglich gespeicherten Instanz einer benutzerdefinierten Klasse nur ein generisches Objekt zurückgegeben wird. Wenn statt dessen der Original-Datentyp auch beim Auslesen erhalten bleiben soll, muss vor dem Aufruf von `SharedObject.getLocal()` die zugehörige Klasse mit Hilfe der Methode `registerClassAlias()`[54] registriert werden.

7.3 Einschätzung

Mit den neuen APIs ermöglicht HTML5 die Schwächen von HTTP-Cookies zu überwinden und erlaubt insbesondere durch die Indexed Database API komplexe Datenstrukturen aufzubauen,

[53] http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager07.html (abgerufen am 18.06.2011)

[54] http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/package.html#registerClassAlias() (abgerufen am 13.06.2011)

welche über Browsersitzungen hinweg nutzbar sind. Die Web Storage API sticht dagegen durch ihre einfache Benutzung und weite Verbreitung hervor. Dem gegenüber stehen die Local Shared Objects in Adobe Flex mit den Möglichkeiten, Speicher zwischen verschiedenen Browsern gemeinsam zu nutzen und komplexe Objekte zu speichern. Die fehlende Integration in die Speicherverwaltung der Browser ist allerdings als Kritikpunkt hervorzuheben, da so eine einfache Kontrolle der gespeicherten Informationen erschwert wird.

8 Zusammenfassung und Ausblick

8.1 Fazit

Mit HTML5 und den damit einher gehenden Elementen und APIs hat die Hypertext Markup Language von ihren spärlichen Anfängen als textorienterte Strukturierungssprache einen riesigen Schritt nach vorn in ein Web mit reichhaltigen und multimedialen Inhalten gewagt. Dieser Schritt ist gelungen, denn die verschiedenen Browserhersteller orientieren sich nicht nur vermehrt an dieser Spezifikation, sondern auch die Spezifikation daran, was im Web wirklich genutzt und gebraucht wird. Hervorzuheben ist, dass selbst Microsoft die Zeichen erkannt und eine Generalüberholung des hauseigenen Browsers angestoßen hat. Ein Hauptaugenmerk liegt hierbei auf der Unterstützung von HTML5. Bedauerlich ist jedoch, dass einige für den HTML5/JavaScript-Prototypen essentielle Teile wie die CORS-Spezifikation, das XMLHttpRequest Level 2- sowie das `FormData`-Objekt noch nicht unterstützt werden. Der HTML5-Prototyp kann daher nicht mit dem Internet Explorer einschließlich Version 9 genutzt werden. Auch die Platform Preview des Internet Explorer 10 verfügt trotz neuer Fähigkeiten nicht über diese APIs [IE10PP].

Nachfolgend eine Zusammenfassung und Bewertung der einzelnen Teilaspekte des Prototypen einer Medienverwaltung in Bezug auf Implementierung und Konzeption der dafür verwendeten Techniken in HTML5/JavaScript und Adobe Flex:

Teilaspekt	HTML5/JavaScript	Adobe Flex
Umsetzung der grafischen Bedienoberfläche	2	1
Einbindung von Quellen	1	3
Drag-and-Drop von Ressourcen & Dateien	2	3
Ansicht von Ressourcen	2	1
Sitzungsübergreifende Persistierung von Daten	1	1
DURCHSCHNITT	1,6	1,8

Tabelle 8.1: Gesamtbewertung im Schulnotensystem

Während das strukturelle Grundgerüst des HTML5-Prototypen dank Formatierungen per CSS und Programmlogik mittels JavaScript zu einer ansprechenden Oberfläche umfunktioniert werden kann, wird jedoch ersichtlich, dass dies nicht dem ursprünglichen Konzept von HTML entspricht. Der Entwicklungsaufwand gestaltet sich daher höher als bspw. bei Adobe Flex, welches explizit auf die Erstellung grafischer Applikationen inkl. Bedienelemente ausgelegt ist. Durch die Verwendung von UI-Bibliotheken unter Verwendung von CSS und JavaScript ließe sich dies in HTML5 allerdings relativieren. Eine Wiederverwendbarkeit und schnelle Entwicklung ist damit ebenso möglich. Auch bei der Gestaltung des Prototypen bietet Adobe Flex durch die Wechselwirkung von Stilen, Skins und Layouts mehr Möglichkeiten als HTML5.

Zustände von Applikationen und Bedienelementen und Übergänge sind allerdings bei beiden vergleichbar zu nutzen.

Beim Zugriff auf HTTP- und WebSocket-Quellen zeigen sich in HTML5/JavaScript und Adobe Flex hauptsächlich Unterschiede im Detail, wobei diese mitunter deutlich sichtbar sind. So ist die grundsätzliche Kommunikation mittels HTTP in beiden Implementierungen vergleichbar, bei der Integration von Datei-Übertragungen allerdings gibt es z.T. Einschränkungen in Adobe Flex. Nicht gemeinsam multiple Dateien und Angaben zur damit durchzuführenden Aktion auf einmal übertragen zu können, stellt insbesondere in Bezug auf die Anforderungen an den Prototypen eine Einschränkung dar. Umwege müssen bestritten werden, ohne allerdings die Funktionalität in HTML5/JavaScript damit erreichen zu können. In Bezug auf die Kommunikation über die Grenzen einer Domain hinweg hat HTML5 mit den Möglichkeiten in Adobe Flex gleichgezogen. Einer weitreichenden Verbindung verschiedener Web-Applikationen und Webservices, wie am Beispiel der Integration fremder Quellen im Prototypen zu sehen, steht damit kaum noch etwas im Wege. Auch die Option von Verbindungen per Sockets ist nun in HTML5 in Form der WebSocket-API gegeben und damit vergleichbar mit den Möglichkeiten in Adobe Flex. Leichtgewichtige und jederzeit von den Kommunikationspartnern eigenständig ausgelöste Datenübertragungen in jede Richtung sind nunmehr möglich. Ob Online-Spiele, gemeinsames Entwickeln oder ressourcensparende Kommunikationskanäle zwischen Geschäftsschnittstellen, Bedarf gibt es genug.

Auch die Unterstützung der bisher auf Desktops beschränkten Interaktion per Drag-and-Drop ist nunmehr im Web nutzbar. Für die Verwaltung von Ressourcen zwischen verschiedenen Quellen im Prototyp ist diese Form der Interaktion wie geschaffen. Die direkte Integration von Dateien auf diesem Wege ist in HTML5 beispiellos und mit den eingeschränkten Möglichkeiten in Adobe Flex nicht zu vergleichen. Ein Drag-and-Drop von Dateien vom lokalen Speicher des Nutzers in den Prototypen ist darin schlichtweg unmöglich. In Bezug auf die Handhabung von Drag-and-Drop-Ereignissen hat Adobe Flex allerdings eine besser nachvollziehbare API und ein simples Standardverhalten lässt sich für listenbasierte Komponenten weitaus schneller einrichten. Sollen jedoch Dateien vom lokalen Speicher bezogen und per Drag-and-Drop verarbeitet werden, geht an HTML5 kein Weg vorbei. Es ist zu hoffen, dass bei diesem allerdings noch Verbesserungen an der API vorgenommen werden.

Der Wunsch, für die Integration von Inhalten wie Audio und Video ohne Plugins wie Adobes Flash auskommen zu können, wurde erhört und die Spezifikation in HTML5 erfreut sich bereits einer weiten Verbreitung. Nicht zuletzt angesichts der Tatsache, dass die Integration des Flash-Plugins in mobile Endgeräten wie dem iPhone von Apple nicht immer möglich oder gewünscht ist, erlauben die neuen Media-Elemente plattformübergreifende Lösungen. Bei den gestalterischen Möglichkeiten für die Bedienelemente von Audio- und Videowiedergabe wird mit den entwickelten Prototypen deutlich, dass HTML5 hier den Vergleich mit Adobe Flex nicht scheuen muss. Problematisch ist jedoch die Vielzahl der von Applikations-Entwicklern

zu unterstützenden Formate und Codecs. Ein Vorhalten und ggf. Umwandeln von Inhalten in diese ist daher für eine breite Unterstützung von Browsern unumgänglich. Bei Adobe Flex kann dieser Aspekt bedingt durch die stets gleiche Laufzeitumgebung in Form des Flash-Players vernachlässigt werden. Freie Implementierungen des SWF-Formats wie Gnash [Gnash] und Lightspark [Lightspark] unterstützen die gleichen Formate wie der proprietäre Flash-Player. Das in der Flex-Implementierung des Prototypen verwendete Open Source Media Framework erlaubt zudem prinzipiell die Unterstützung beliebiger Formate und Codecs durch seine Plugin-API. In HTML5 gibt es keine Möglichkeit der funktionellen Erweiterung durch den Entwickler einer Web-Applikation, weshalb man hier gänzlich auf die Implementierungen durch die Browser-Entwickler angewiesen ist.

Die mit der Web Storage API und der Indexed Database API in HTML5/JavaScript eingeführten Möglichkeiten zur lokalen Speicherung von Informationen über die Grenzen einer Browsersitzung hinweg überflügeln althergebrachte HTTP-Cookies bei weitem. Von simplen bis hin zu strukturierten Informationen sind diese eine Bereicherung in Hinsicht auf eine erweiterte lokale Datenverwaltung. Dahingehend muss sich Adobe Flex allerdings nicht verstecken, da dessen Local Shared Objects (LSO) mit beliebigen Daten umgehen kann. Eine vergleichbare Persistierung beliebiger Objekten ließe sich so in HTML5/JavaScript nur mittels Serialisierung z.B. per JSON umsetzen. In Adobe Flex ist diese Funktionalität durch das Action Message Format standardmäßig gegeben. Bedingt durch die Natur der LSO können in Flex zudem Daten zwischen verschiedenen Clients geteilt werden, was sich je nach Applikation auch als hilfreiche Eigenschaft erweisen kann.

Zusammenfassend lässt sich sagen, dass HTML5 nicht nur reif für die Nutzung sondern bei näherer Betrachtung bereits allgegenwärtig ist [CanIUse]. Beispiele wie Youtube für die Wiedergabe von Video [Youtube-HTML5], Google Mail für die Integration von Drag-and-Drop zum Anhängen von Dateien [GoogleMail-HTML5] oder GTK für die Integration eines HTML5-Backends [GTK3-HTML5] zeigen, dass es sich lohnt, die Fülle der neuen Möglichkeiten näher zu betrachten und damit den eigenen Horizont zu erweitern.

Viele Teile der Spezifikation können bedenkenlos auf jedes HTML-Dokument angewandt werden. Dies gründet darauf, dass diese Teile schlichtweg darauf basieren, was bisher im Web durch die ungenauen Regeln zum Parsen von HTML-Dokumenten alltäglich war, wie z.B. die Abkürzung der Zeichenkodierung eines Dokuments mittels `meta`-Element. Ein anderes Beispiel ist die Verwendung der neuen Feldtypen für Formulare; Browser, die diese nicht verstehen, stellen sie einfach als einfache Textfelder dar, die Nutzbarkeit bleibt gewährleistet. Andere Teile sind unterschiedlich emulierbar, wie bspw. die Nachbildung der besagten Feldtypen mittels JavaScript. Und einige Teile erfordern die Unterstützung durch einen aktuellen Browser; hierbei stellt die Wiedergabe von Audio- und Video ein gutes Beispiel dar. Mathias Bynes stellt dies in seinem 3-Level-System anschaulich dar [Bynens10].

8.2 Ausblick

Wie an der Entscheidung der WHATWG erkennbar, die Versionsnummer aus der Spezifikation zu entfernen, handelt es sich bei HTML um eine lebendige Spezifikation. Sie wächst mit den Anforderungen im Web und stellt eine Plattform für die Verbreitung kreativer Ideen zur Verfügung. Neben unvollständigen Implementierungen bedürfen daher auch Teile der Spezifikation noch Revisionen für eine flächendeckende Verbreitung. In der HTML-Spezifikation der WHATWG [HTML-WHATWG] werden praktisch täglich Änderungen eingepflegt und Korrekturen vorgenommen. In Form von festen Versionen der HTML-Spezifikation wird das W3C auch in Zukunft einen Entwicklungsstand festhalten und als Empfehlung zur Implementierung freigeben. Das schnelle und lebendige Entwicklungsmodell der WHATWG soll hierbei nicht abschrecken sondern zur Teilnahme anregen.

Durch die umfassenden Möglichkeiten von HTML5 verschwindet die Grenze zwischen Desktop- und Web-Applikation immer weiter. Die HTML-Spezifikation wird immer mehr Funktionen, die früher nur in Desktop-Applikationen oder mit Hilfe von Plugins nutzbar waren, integrieren. Als Beispiele seien hier die Integration von Videokonferenzen und Peer-to-Peer-Kommunikation genannt; zudem die Arbeit der Device APIs and Policy Working Group [DAP] mit dem Ziel, Web-Applikationen Zugriff auf Elemente wie Kalender, Adressbücher, Videokameras, Batteriezustand und Netzwerkverbindung zu ermöglichen. Immer mehr per HTML5 und JavaScript entwickelte Applikation werden auf dem Desktop einziehen, wie z.B. an den Plänen Microsofts für eine neue Bedienoberfläche in seinem Betriebssystem Windows 8 erkennbar ist [Bright11]. Mit dem Aufkommen immer mehr Spezifikationen unter dem Mantel HTML(5) ist absehbar, dass reine Desktop-Applikationen im traditionellen Sinn an Bedeutung verlieren werden. Der „Web-Desktop“ kann Realität werden, wie bereits anhand von Googles Chromium OS sichtbar ist [ChromiumOS]. Unvermeidlich hierfür ist jedoch, dass der flächendeckende Ausbau von Breitbandverbindungen vorangetrieben wird.

Adobe Flex und das Flash-Plugin werden nicht plötzlich von der Bildschirmfläche verschwinden, dafür bieten sie eine viel zu gut durchdachte Entwicklungsumgebung für komplexe Applikationen. Dennoch wird HTML5 verstärkt Bereiche einnehmen, die bisher Adobe Flex und anderen Flash-Frameworks vorbehalten waren und das Flash-Plugin als erste Wahl für erweiterte Funktionen dahingehend immer mehr in Frage stellen. Wenn eine native Implementierung mittels HTML5 der eines Plugins wie Flash gegenüber steht, was grundsätzlich als Fremdkörper im Browser agiert, wird immer häufiger die Entscheidung zugunsten ersterem fallen. Aber während das Flash-Plugin bereits seit langem mit Hardware-Beschleunigung aufwarten kann, muss diese sich im Kontext HTML erst noch etablieren. Grafisch aufwändige Applikationen werden daher in absehbarer Zeit Adobe Flex und dem Flash-Plugin vorbehalten sein.

Ein absehbares Ziel des Flex-Frameworks von Adobe besteht darin, das Defizit im mobilen Bereich durch eine aktive Unterstützung von mehr Geräten zu kompensieren. Diese Entwicklung ist durch die letzten Versionen des Flex-SDK ersichtlich [Flex-Mobile]. Trotz der Quelloffenheit des Flex-Frameworks ist jedoch die von Adobe geplante Zukunft derzeit nicht absehbar. Die Integration einer neuen Version des Open Source Media Frameworks dürfte darin allerdings enthalten sein. Inwieweit die im Zuge dieser Arbeit aufgelisteten Beschränkungen aufgehoben oder kompensiert werden, muss dagegen abgewartet werden.

Inwieweit die unterschiedlichen Entwicklungsprozesse von HTML5 und Adobe Flex in Zukunft zum Tragen kommen, muss beobachtet werden. Während bei HTML5 der Kreis der beteiligten Entwickler mit Mozilla, Google, Apple, Microsoft und vielen unabhängigen Entwicklern sehr groß ist, wird die zukünftige Entwicklung des Flex-SDK faktisch nur Adobe beeinflusst. Eine Inspiration und Motivation zum Wettbewerb durch andere Teilnehmer ist dadurch nicht gegeben. Durch die Öffnung des Quellcodes von Flex durch Adobe konnte dieser Nachteil zwar z.T. ausgeglichen werden, die Entscheidung, welche Features und Fehlerkorrekturen in das SDK einfließen, obliegt am Ende aber weiterhin einer Instanz allein. In HTML5 gestaltet sich die Situation anders und potentielle Probleme durch Eigenentwicklungen von Browserherstellern lösen sich durch den Wettbewerb und Selbstregulierung.

Schließlich verfolgen alle das gleiche Ziel: das Web mit reichhaltiger Funktionalität immer mehr Menschen zugänglich zu machen.

Anhang

MediaRequest-Protokoll-Syntax

```
Ziffer = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
Zeichen = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" |
          "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" |
          "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" |
          "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;
Alphanumerische Zeichen = Ziffer | Zeichen ;
Prüfsummen Zeichen = Ziffer | "a" | "b" | "c" | "d" | "e" | "f" ;
SHA1-Prüfsumme = 40 * Prüfsummen Zeichen
Base64 Zeichen = Alphanumerische Zeichen | "+" | "/" ;
Version = Ziffer, {Ziffer}, ".", Ziffer, {Ziffer} ;
Typ = "info" | "list" | "add" | "remove" ;
Bezeichner = Alphanumerische Zeichen, { Alphanumerische Zeichen } ;
Wert = <Unicode-Zeichen>* ;
Ressource = <URL> ":" "{"
              [ Bezeichner ":" Wert ]
              { "," Bezeichner ":" Wert }
            "}" ;
Ressourcenliste = "{" { Ressource } "}" ;
Datei = "{"
          "name" ":" Bezeichner, ".", Bezeichner ","
          "type" ":" Bezeichner, "/", Bezeichner ","
          "size" ":" Ziffer, { Ziffer } ","
          "data" ":" Base64 Zeichen, { Base64 Zeichen } ","
        "}" ;
Dateiliste = "{" { Datei } "}" ;
Zugang = "{" "username" ":" Bezeichner "," "password" ":" SHA1-Prüfsumme "}" ;
Anfrage = "mediaRequest" "{"
            "version" ":" Version ","
            "type" ":" Typ
            [ "," "resources" ":" Ressourcenliste ]
            [ "," "files" ":" Dateiliste ]
            [ "," "credentials" ":" Zugang ]
          "}" ;
Ergebnis = "success" | "error" ;
Quell-Information = "{"
                      [ "title" ":" <Unicode-Zeichen>* ]
                      [ "," "allowed" ":" "[" Typ { "," Typ } "]" ]
                      [ "," "restricted" ":" "[" Typ { "," Typ } "]" ]
                    "}" ;
Fehlercode = "unknown" | "unknownProtocol" | "unknownProtocolVersion" | "unknownRequestType" |
             "connectionFailed" | "disconnected" | "authenticationRequired" |
             "authenticationFailed" | "malformedRequest" | "malformedResponse" ;
Fehler = "{" "code" ":" Fehlercode [ "," "message" ":" <Unicode-Zeichen>* ] "}" ;
Antwort = "{"
            "version" ":" Version ","
            "type" ":" Typ ","
            "result" ":" Ergebnis
            [ "," "info" ":" Quell-Information ]
            [ "," "resources" ":" Ressourcenliste ]
            [ "," "error" ":" Fehler ]
          "}" ;
```

Listing A: EBNF-Beschreibung des MediaRequest-Protokolls

Prototypen-Klassendiagramm

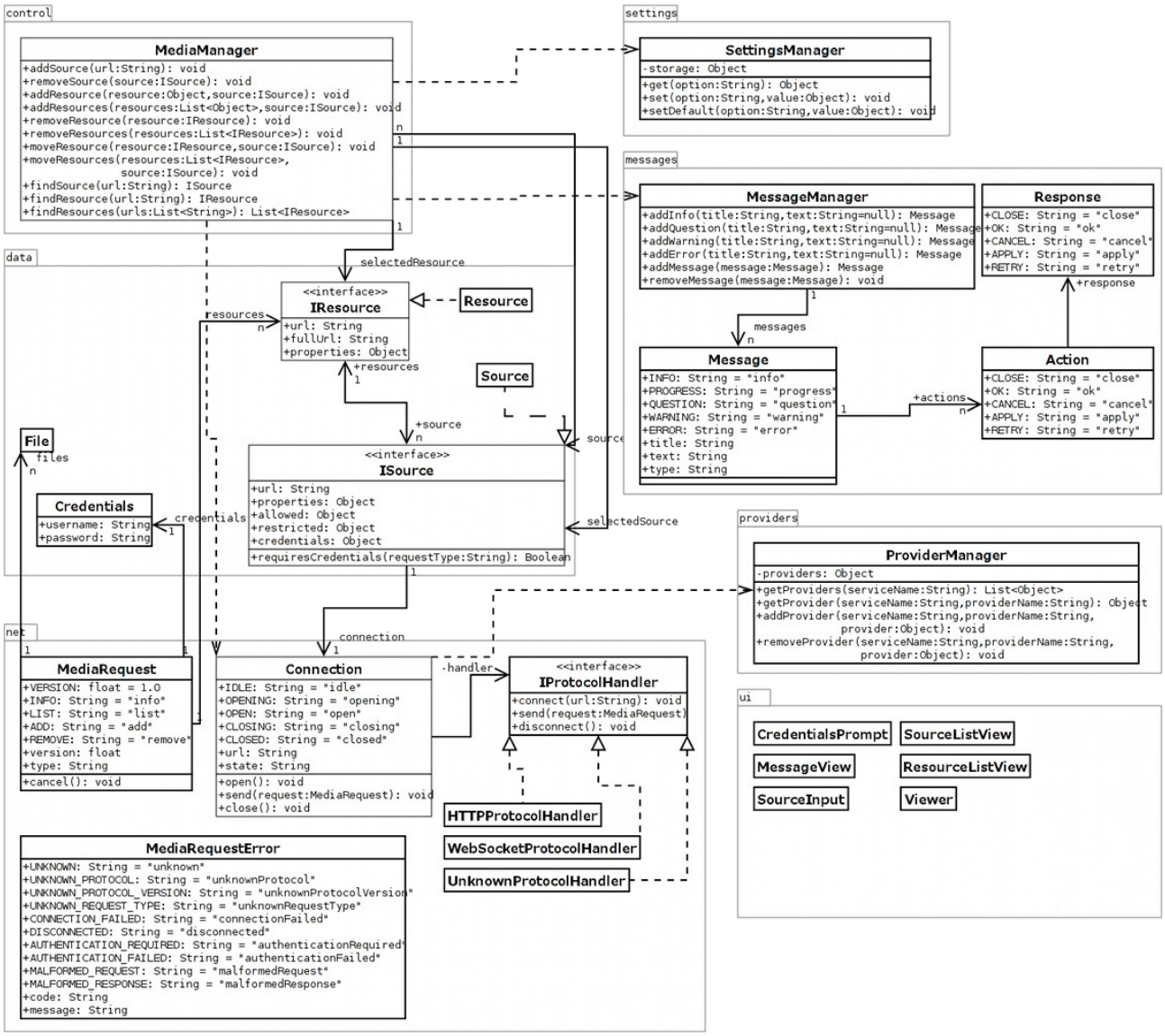

Abbildung A: Klassendiagramm des Prototypen

Literaturverzeichnis

[Adobe] Adobe, *Adobe Website*, 2011, http://www.adobe.com/ (abgerufen am 19.06.2011)

[AIR] Adobe, *Adobe Integrated Runtime*, 2011, http://www.adobe.com/de/products/air/ (abgerufen am 19.06.2011)

[Almaer07] Dion Almaer, *JSON vs. XML: The Debate*, 2007, http://ajaxian.com/archives/json-vs-xml-the-debate (abgerufen am 19.06.2011)

[Almaer10] Dion Almaer, *fullscreen API coming to browsers near you?*, 2010, http://ajaxian.com/archives/fullscreen-api-coming-to-browsers-near-you (abgerufen am 19.06.2011)

[AS3] G. Grossman, E. Huang, *ActionScript 3.0 overview*, 2006, http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html (abgerufen am 19.06.2011)

[AS3corelib] Mike Chambers, *An ActionScript 3 Library that contains a number of classes and utilities for working with ActionScript 3.*, 2011, https://github.com/mikechambers/as3corelib (abgerufen am 21.06.2011)

[AudioDataAPI] Mozilla Foundation, *Audio Data API*, 2011, https://wiki.mozilla.org/Audio_Data_API (abgerufen am 19.06.2011)

[Blizzard10] Christopher Blizzard, *disabling websockets for firefox 4*, 2010, http://www.0xdeadbeef.com/weblog/2010/12/disabling-websockets-for-firefox-4/ (abgerufen am 19.06.2011)

[Bright11] Peter Bright, *Microsoft gives the first official look of Windows 8 touch interface*, 2011, http://arstechnica.com/microsoft/news/2011/06/microsoft-gives-the-first-official-look-of-windows-8-touch-interface.ars (abgerufen am 19.06.2011)

[Bynens10] Mathias Bynens, *The three levels of HTML5 usage*, 2010, http://mathiasbynens.be/notes/html5-levels (abgerufen am 19.06.2011)

[CanIUse] Alexis Deveria, *When can I use... Support tables for HTML5, CSS3, etc*, 2011, http://caniuse.com/ (abgerufen am 19.06.2011)

[Canvas2D] Ian Hickson, *HTML Canvas 2D Context*, 2011, http://www.w3.org/TR/2011/WD-2dcontext-20110405/ (abgerufen am 19.06.2011)

[CanvasDemos] Nested Elements, *Applications, games, tools and tutorials for the HTML5 canvas element - Canvas Demos*, 2011, http://www.canvasdemos.com/ (abgerufen am 19.06.2011)

[Chambers07] Mike Chambers, *AIR Example : Native Drag and Drop*, 2007, http://www.mikechambers.com/blog/2007/11/07/air-example-native-drag-and-drop/ (abgerufen am 19.06.2011)

[ChromeDevTools] Google, *Google Chrome Developer Tools*, 2011, http://code.google.com/intl/de/chrome/devtools/ (abgerufen am 19.06.2011)

[ChromiumOS] Google, *Chromium OS*, 2009, http://www.chromium.org/chromium-os (abgerufen am 21.06.2011)

[CORS] Anne van Kesteren, *Cross-Origin Resource Sharing*, 2010, http://www.w3.org/TR/2010/WD-cors-20100727/Anne van Kesteren (abgerufen am

19.06.2011)

[CSS] W3C, *Cascading Style Sheets*, 2011, http://www.w3.org/Style/CSS/ (abgerufen am 19.06.2011)

[CSSCurrent] W3C, *CSS current work*, 2011, http://www.w3.org/Style/CSS/current-work (abgerufen am 19.06.2011)

[CSSIE] Microsoft, *CSS Compatibility and Internet Explorer*, 2011, http://msdn.microsoft.com/en-us/library/cc351024(VS.85).aspx (abgerufen am 19.06.2011)

[CSSMozilla] Mozilla Developer Network, *Mozilla CSS support chart*, 2011, https://developer.mozilla.org/en/Mozilla_CSS_support_chart (abgerufen am 19.06.2011)

[CSSOpera] Opera ASA, *Web specifications support in Opera Presto 2.8*, 2011, http://www.opera.com/docs/specs/presto28/ (abgerufen am 19.06.2011)

[CSSSafari] Apple, *Safari CSS Reference*, 2011, http://developer.apple.com/library/safari/documentation/appleapplications/reference/SafariCSSRef/Introduction.html (abgerufen am 19.06.2011)

[DAP] W3C, *Device APIs and Policy Working Group*, 2011, http://www.w3.org/2009/dap/ (abgerufen am 19.06.2011)

[DOM] W3C, *Document Object Model*, 2005, http://www.w3.org/DOM/ (abgerufen am 19.06.2011)

[DSP-JS] Corban Brook, *Digital Signal Processing for Javascript*, 2010, https://github.com/corbanbrook/dsp.js (abgerufen am 19.06.2011)

[EBNF] ISO/IEC, *Extended Backus-Naur Form*, 1996, http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf (abgerufen am 19.06.2011)

[ECMA262-1] ECMA, *ECMAScript: A general purpose, cross-platform programming language*, 1997, http://www.ecmascript.org/docs.php (abgerufen am 19.06.2011)

[ExCanvas] Google, *HTML5 Canvas for Internet Explorer*, 2011, http://code.google.com/p/explorercanvas/ (abgerufen am 19.06.2011)

[FB] Adobe, *Adobe Flash Builder*, 2011, http://www.adobe.com/de/products/flash-builder.html (abgerufen am 19.06.2011)

[FFmpeg] FFmpeg project, *FFmpeg*, 2011, http://ffmpeg.org/ (abgerufen am 19.06.2011)

[FileAPI] A. Ranganathan, J. Sicking, *File API Working Draft*, 2010, http://www.w3.org/TR/2010/WD-FileAPI-20101026/ (abgerufen am 19.06.2011)

[FileWriterAPI] Eric Uhrhane, *File API: Writer*, 2011, http://www.w3.org/TR/2011/WD-file-writer-api-20110419/ (abgerufen am 19.06.2011)

[Firebug] Mozilla Foundation, *Firebug*, 2011, http://getfirebug.com/ (abgerufen am 19.06.2011)

[Flash10-UIA] Ian Melven, *User-initiated action requirements in Flash Player 10*, 2008, http://www.adobe.com/devnet/flashplayer/articles/fplayer10_uia_requirements.html (abgerufen am 19.06.2011)

[Flex-Mobile] Narciso Jaramillo, *Mobile development using Adobe Flex 4.5 SDK and Flash Builder 4.5*, 2011, http://www.adobe.com/devnet/flex/articles/mobile-development-flex-flashbuilder.html (abgerufen am 21.06.2011)

[Flex03] Macromedia, *Macromedia Introduces Macromedia Flex Product Strategy*, 2003, http://www.adobe.com/macromedia/proom/pr/2003/introducing_flex.html (abgerufen am 19.06.2011)

[Flex11] Adobe, *Flex SDK*, 2011, http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK (abgerufen am 19.06.2011)

[FlexBindable] Adobe, *Using the Bindable metadata tag*, 2011, http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf69084-7cc5.html (abgerufen am 19.06.2011)

[FlexCSS] Adobe, *Using Cascading Style Sheets*, 2011, http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf62883-7ff2.html (abgerufen am 19.06.2011)

[FlexCSSProps] Frankie Loscavio, *Flex 3.0 CSS Properties List*, 2011, http://www.loscavio.com/downloads/blog/flex3_css_list/flex3_css_list.htm (abgerufen am 19.06.2011)

[FlexDataBinding] Adobe, *About data binding*, 2011, http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf64c3d-7fff.html (abgerufen am 19.06.2011)

[FlexSkins] Adobe, *About Spark skins*, 2011, http://help.adobe.com/en_US/flex/using/WSC8DB0C28-F7A6-48ff-9899-7957415A0A49.html (abgerufen am 19.06.2011)

[FullscreenAPI] Mozilla Foundation, *Gecko FullScreenAPI*, 2011, https://wiki.mozilla.org/index.php?title=Gecko:FullScreenAPI (abgerufen am 19.06.2011)

[Gnash] Free Software Foundation, *GNU Gnash*, 2005, http://www.gnu.org/s/gnash/ (abgerufen am 19.06.2011)

[GoogleMail-HTML5] Google, *Drag and drop attachments onto messages*, 2010, http://gmailblog.blogspot.com/2010/04/drag-and-drop-attachments-onto-messages.html (abgerufen am 19.06.2011)

[GTK-DnD] Ryan McDougall, *Drag and Drop with GTK+*, 2005, http://live.gnome.org/GnomeLove/DragNDropTutorial (abgerufen am 21.06.2011)

[GTK3-HTML5] Alexander Larsson, *Gtk3 vs HTML5*, 2010, http://blogs.gnome.org/alexl/2010/11/23/gtk3-vs-html5/ (abgerufen am 19.06.2011)

[Hopmann07] Alex Hopmann, *The story of XMLHTTP*, 2007, http://www.alexhopmann.com/xmlhttp.htm (abgerufen am 19.06.2011)

[HTML-WHATWG] WHATWG, *HTML Living Standard*, 2011, http://www.whatwg.org/specs/web-apps/current-work/multipage/ (abgerufen am 19.06.2011)

[HTML1] Tim Berners-Lee, *HyperText Mark-up Language*, 1992, http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/MarkUp.html (abgerufen am 19.06.2011)

[HTML2] Tim Berners-Lee, *Hypertext Markup Language - 2.0*, 1995, http://tools.ietf.org/html/rfc1866 (abgerufen am 19.06.2011)

[HTML4] W3C, *HTML 4.01 Specification*, 1999, http://www.w3.org/TR/1999/REC-html401-19991224/ (abgerufen am 19.06.2011)

[HTML5] W3C, *HTML5 Working Draft*, 2011, http://www.w3.org/TR/2011/WD-html5-20110525/ (abgerufen am 19.06.2011)

[HTML5-DnD] Ian Hickson, *HTML5 – Drag and drop*, 2011, http://www.w3.org/TR/2011/WD-html5-20110525/dnd.html (abgerufen am 19.06.2011)

[HTML5Canvas] W3C, *The canvas element*, 2011, http://www.w3.org/TR/2011/WD-html5-20110525/the-canvas-element.html (abgerufen am 19.06.2011)

[HTML5FormData] F. de Metz, A. ONeal, *Emulate FormData object for some browsers*, 2011, https://github.com/coolaj86/html5-formdata (abgerufen am 19.06.2011)

[Hype] Tumult Inc., *Tumult Hype*, 2011, http://tumultco.com/hype/ (abgerufen am 19.06.2011)

[ID3] M. Nilsson, M. Mutschler, J. Sundstrom, u.a., *ID3 is a [...] audio file data tagging format [...]*, 2011, http://www.id3.org/ (abgerufen am 21.06.2011)

[IE10PP] Microsoft, *Native HTML5: First IE10 Platform Preview Available for Download*, 2011, http://blogs.msdn.com/b/ie/archive/2011/04/12/native-html5-first-ie10-platform-preview-available-for-download.aspx (abgerufen am 21.06.2011)

[IE9Canvas] Microsoft, *Internet Explorer 9 Guide for Developers - HTML5 canvas Element*, 2011, http://msdn.microsoft.com/en-us/ie/ff468705#_HTML5_canvas (abgerufen am 19.06.2011)

[IEDocMode] Marc Silbey, *How IE8 Determines Document Mode*, 2010, http://blogs.msdn.com/b/ie/archive/2010/03/02/how-ie8-determines-document-mode.aspx (abgerufen am 19.06.2011)

[IndexedDB] Nikunj Mehta, Jonas Sicking, Eliot Graff, Andrei Popescu, Jeremy Orlow, *Indexed Database API*, 2011, http://www.w3.org/TR/2011/WD-IndexedDB-20110419/ (abgerufen am 19.06.2011)

[JQuery] The jQuery Project, *jQuery: The Write Less, Do More, JavaScript Library*, 2011, http://jquery.com/ (abgerufen am 19.06.2011)

[JQueryUI] The jQuery Project, *jQuery user interface*, 2011, http://jqueryui.com/ (abgerufen am 19.06.2011)

[JSON] Douglas Crockford, *JavaScript Object Notation*, 2006, http://json.org/ (abgerufen am 19.06.2011)

[JSON-JS] Douglas Crockford, *JSON in JavaScript*, 2011, https://github.com/douglascrockford/JSON-js (abgerufen am 19.06.2011)

[Lightspark] Alessandro Pignotti, *Lightspark*, 2011, http://lightspark.sourceforge.net/ (abgerufen am 19.06.2011)

[Lisci10] Marco Lisci, *HTML5 Canvas Image Effects: Black & White*, 2010, http://spyrestudios.com/html5-canvas-image-effects-black-white/ (abgerufen am 19.06.2011)

[MDCFormData] Mozilla Developer Network, *Using XMLHttpRequest – Using FormData objects*, 2011, https://developer.mozilla.org/En/XMLHttpRequest/Using_XMLHttpRequest#Using_FormData_objects (abgerufen am 19.06.2011)

[MooTools] The MooTools Dev Team, *MooTools - a compact javascript framework*, 2011,

http://mootools.net/ (abgerufen am 19.06.2011)

[MS-DnD] Microsoft, *About DHTML Data Transfer*, 2007, http://msdn.microsoft.com/en-us/library/ms537658(VS.85).aspx (abgerufen am 21.06.2011)

[MXML-AS11] Adobe, *Programmieren mit MXML und ActionScript*, 2011, http://www.adobe.com/de/devnet/flex/quickstart/coding_with_mxml_and_actionscript/ (abgerufen am 19.06.2011)

[Nadel09] Ben Nadel, *Default To The Numeric, Email, And URL Keyboards On The iPhone*, 2009, http://www.bennadel.com/blog/1721-Default-To-The-Numeric-Email-And-URL-Keyboards-On-The-iPhone.htm (abgerufen am 19.06.2011)

[Ochard09] Leslie Michael Ochard, *Drag and Drop in Firefox 3.5*, 2009, http://decafbad.com/2009/07/drag-and-drop/api-demos.html (abgerufen am 19.06.2011)

[Ødegaard10] Ruarí Ødegaard, Opera ASA, *Friday morning improvements*, 2010, http://my.opera.com/desktopteam/blog/2010/12/10/friday-morning-improvements (abgerufen am 19.06.2011)

[Opera-GStreamer] Opera ASA, GStreamer team, *GStreamer source code - Opera*, 2011, http://sourcecode.opera.com/gstreamer/ (abgerufen am 19.06.2011)

[OperaWSC] Opera ASA, *Opera Web Standards Curriculum*, 2011, http://www.opera.com/company/education/curriculum/ (abgerufen am 19.06.2011)

[OSMF] Adobe, *Open Source Media Framework*, 2011, http://www.opensourcemediaframework.com/ (abgerufen am 19.06.2011)

[Ranganathan09] Arun Ranganathan, *cross-site xmlhttprequest with CORS*, 2009, http://hacks.mozilla.org/2009/07/cross-site-xmlhttprequest-with-cors/ (abgerufen am 19.06.2011)

[Remi11] Lous Remi, *Aurora 6 is here*, 2011, http://hacks.mozilla.org/2011/05/aurora-6-is-here/ (abgerufen am 19.06.2011)

[ResetCSS] Richard Clark, *HTML5 Reset Stylesheet*, 2010, http://html5doctor.com/html-5-reset-stylesheet/ (abgerufen am 19.06.2011)

[RFC2388] L. Masinter, *Returning Values from Forms: multipart/form-data*, 1998, http://www.faqs.org/rfcs/rfc2388.html (abgerufen am 19.06.2011)

[RFC2616] T. Berners-Lee, R. Fielding, J. Gettys, u.a., *Hypertext Transfer Protocol -- HTTP/1.1*, 1999, http://tools.ietf.org/html/rfc2616 (abgerufen am 21.06.2011)

[RFC3797] Larry Masinter, *The "data" URL scheme*, 1998, http://tools.ietf.org/html/rfc2397 (abgerufen am 21.06.2011)

[RFC4648] S. Josefsson, *The Base16, Base32, and Base64 Data Encodings*, 2006, http://tools.ietf.org/html/rfc4648 (abgerufen am 19.06.2011)

[RFC6265] Adam Barth, *HTTP State Management Mechanism*, 2011, http://tools.ietf.org/html/rfc6265 (abgerufen am 19.06.2011)

[Russel06] Alex Russell, *Comet: Low Latency Data for the Browser*, 2006, http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/ (abgerufen am 19.06.2011)

[Schäfer11] Mathias Schäfer, *JavaScript: Entstehung und Standardisierung*, 2011,

http://molily.de/js/standards.html#netscape-microsoft (abgerufen am 19.06.2011)

[SharedObject] Macromedia, *Macromedia Flash - ActionScript dictionary: SharedObject*, 2011, http://www.adobe.com/support/flash/action_scripts/actionscript_dictionary/actionscript_dictionary648.html (abgerufen am 19.06.2011)

[SOP] W3C, *Same-Origin Policy*, 2011, http://www.w3.org/Security/wiki/Same_Origin_Policy (abgerufen am 19.06.2011)

[Stead09] Mike Stead, *Upload Multiple Files With a Single Request in Flash*, 2009, http://blog.mikestead.me/upload-multiple-files-with-a-single-request-in-flash/ (abgerufen am 19.06.2011)

[Subramaniam10] Deepa Subramaniam, *A brief overview of the Spark architecture and component set*, 2010, http://www.adobe.com/devnet/flex/articles/flex4_sparkintro.html (abgerufen am 21.06.2011)

[SWFHR] Jim R. Wilson, *SWFHttpRequest*, 2007, http://jimbojw.com/wiki/index.php?title=SWFHttpRequest_Flash/Ajax_Utility (abgerufen am 19.06.2011)

[TxWebSocket] R. Lotun, C. Simpson, *Twisted WebSocket Server*, 2011, https://github.com/rlotun/txWebSocket (abgerufen am 19.06.2011)

[VideoParameters] WHATWG, *Video type parameters*, 2011, http://wiki.whatwg.org/wiki/Video_type_parameters (abgerufen am 19.06.2011)

[W3C] W3C, *World Wide Web Consortium*, 2011, http://www.w3.org/ (abgerufen am 19.06.2011)

[W3SchoolsStats] W3Schools, *Browser Statistics*, 2011, http://www.w3schools.com/browsers/browsers_stats.asp (abgerufen am 19.06.2011)

[Warden08] Jesse Warden, *Flash Player 10 Surprise: Error #2176*, 2008, http://jessewarden.com/2008/10/flash-player-10-surprise-error-2176.html (abgerufen am 19.06.2011)

[WebApps] WHATWG, *Web Applications 1.0*, 2011, http://www.whatwg.org/specs/web-apps/current-work/complete/ (abgerufen am 19.06.2011)

[WebGL] Khronos Group, *WebGL - OpenGL ES 2.0 for the Web*, 2011, http://www.khronos.org/webgl/ (abgerufen am 19.06.2011)

[WebM] Google, *The WebM Project*, 2010, http://www.webmproject.org/ (abgerufen am 19.06.2011)

[WebStorage] Ian Hickson, *Web Storage Working Draft*, 2011, http://www.w3.org/TR/2011/WD-webstorage-20110208/ (abgerufen am 19.06.2011)

[WHATWG] WHATWG, *Web Hypertext Application Technology Working Group*, 2004, http://www.whatwg.org/ (abgerufen am 19.06.2011)

[WS-JS] Hiroshi Ichikawa, *HTML5 Web Socket implementation powered by Flash*, 2011, https://github.com/gimite/web-socket-js (abgerufen am 19.06.2011)

[WSAPI] Ian Hickson, *The WebSocket API*, 2011, http://www.w3.org/TR/2011/WD-websockets-20110419/ (abgerufen am 19.06.2011)

[WSP-00] Ian Hickson, *The WebSocket protocol, handshake version -00*, 2010, http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-00 (abgerufen am 19.06.2011)

[WSP-07] Ian Hickson, *The WebSocket protocol, handshake version -07*, 2011, http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-07 (abgerufen am 19.06.2011)

[WSProtocol] Ian Hickson, *The WebSocket protocol*, 2010, http://www.whatwg.org/specs/web-socket-protocol/ (abgerufen am 19.06.2011)

[XDR] Microsoft, *XDomainRequest Object*, 2011, http://msdn.microsoft.com/en-us/library/cc288060(VS.85).aspx (abgerufen am 19.06.2011)

[XHR1] Anne van Kesteren, *XMLHttpRequest Candidate Recommendation*, 2010, http://www.w3.org/TR/2010/CR-XMLHttpRequest-20100803/ (abgerufen am 19.06.2011)

[XHR2] Anne van Kesteren, *XMLHttpRequest Level 2*, 2010, http://www.w3.org/TR/2010/WD-XMLHttpRequest2-20100907/ (abgerufen am 19.06.2011)

[XHTML1] W3C, *XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) Recommendation*, 2002, http://www.w3.org/TR/2002/REC-xhtml1-20020801/ (abgerufen am 19.06.2011)

[XUAComp] Microsoft, *META Tags and Locking in Future Compatibility*, 2011, http://msdn.microsoft.com/en-us/library/cc817574.aspx (abgerufen am 19.06.2011)

[Youtube-HTML5] Google, *HTML5-Videoplayer von YouTube*, 2011, http://www.youtube.com/html5 (abgerufen am 19.06.2011)

[YUI] Yahoo, *Yahoo User Interface Library*, 2011, http://developer.yahoo.com/yui/ (abgerufen am 19.06.2011)

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit zum Thema

Gegenüberstellung der Technologien HTML5/JavaScript

und des Flex-Frameworks anhand der

prototypischen Umsetzung einer Medienverwaltung

selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt habe.

Dresden, den 27.06.2011

Mathias Brodala