

Hochschule für Technik und Wirtschaft Dresden (FH)

Fachbereich Informatik/Mathematik

# **Diplomarbeit**

im Studiengang Medieninformatik

Thema:

**Prototypische Umsetzung eines Internetportals zur gemeinsamen  
Nutzung von Konsumgegenständen unter Verwendung des MVC-Musters  
mittels Flash und HTML5 zum Vergleich beider Technologien.**

eingereicht von: Marc Frydetzki (20884)

eingereicht am: 28.01.2011

Betreuer: Prof. Dr. Teresa Merino

1	Einleitung.....	1
1.1	Motivation.....	2
1.2	Zielsetzung.....	3
2	Begriffserklärungen.....	4
2.1	Die Geschichte des Internets.....	4
2.2	Browser.....	8
2.3	W3C.....	8
2.4	WHATWG.....	9
2.5	Plattformunabhängigkeit.....	9
2.6	Rich Internet Application.....	10
3	Flash.....	11
3.1	Die Geschichte von Flash.....	11
3.2	ActionScript.....	15
3.2.1	ActionScript 1.0.....	16
3.2.2	ActionScript 2.0.....	16
3.2.3	ActionScript 3.0.....	17
3.3	Dateiformate.....	20
3.3.1	.fla.....	20
3.3.2	.swf.....	20
3.3.3	.as.....	21
3.3.4	.xml.....	21
3.3.5	Weitere Formate.....	22
3.4	Einsatzbereiche.....	23
3.5	Techniken.....	23
3.5.1	Deeplink.....	23
3.5.2	Suchmaschinen-Optimierung.....	24
3.5.3	Dynamisierung.....	25
3.5.4	ExternalInterface.....	25
3.5.5	LocalConnection.....	26
3.5.6	SharedObject.....	26
3.6	Vor- und Nachteile.....	27
3.7	Ausblick.....	28

4 HTML5.....	30
4.1 Die Geschichte von HTML.....	30
4.2 Entwicklungsschritte HTML 1.0 bis XHTML.....	32
4.2.1 HTML 1.0.....	33
4.2.2 HTML 2.0.....	33
4.2.3 HTML 3.0.....	34
4.2.4 HTML 3.2.....	35
4.2.5 HTML 4.0.....	35
4.2.6 HTML 4.01.....	36
4.2.7 XHTML.....	36
4.3 HTML5 – Der aktuelle Stand der Entwicklungen.....	38
4.3.1 Das Canvas-Element.....	40
4.3.2 Das embed-Element.....	40
4.3.3 Die Elemente audio und video.....	41
4.3.4 Die Media-API.....	44
4.3.5 Offline Web Application API.....	45
4.3.6 Semantisches Markup.....	45
4.3.7 Neue Formular Elemente.....	46
4.4 Einsatzbereiche.....	47
4.5 Vor- und Nachteile.....	48
4.6 Ausblick.....	49
5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung.....	51
5.1 Übersicht.....	51
5.1.2 Ziel des Vergleichs.....	53
5.1.3 Nicht betrachtete Aspekte.....	54
5.2 Verwendete Frameworks.....	54
5.2.1 Flash Framework PureMVC.....	55
5.2.1.1 Funktionsumfang.....	57
5.2.1.1 Vorteile.....	59
5.2.1.2 Nachteile.....	59
5.2.2.1 Funktionsumfang.....	62
5.2.2.2 Vorteile.....	67

5.2.2.3 Nachteile.....	67
5.3 Verwendete Entwicklungsumgebungen.....	68
5.3.1 Flash/PureMVC.....	68
5.3.1.1 Vorteile.....	69
5.3.1.2 Nachteile.....	70
5.3.2 SproutCore.....	71
5.3.2.1 Vorteile.....	72
5.3.2.2 Nachteile.....	72
5.3.3 Schlussfolgerung.....	73
5.4 Framework Einstieg/Basisklassen.....	74
5.4.1 Flash/PureMVC spezifisch.....	74
5.4.2 SproutCore spezifisch.....	75
5.5 Implementierung der Models.....	76
5.5.1 PureMVC Models & Proxies.....	76
5.5.2 SproutCore Models.....	83
5.5.3 Schlussfolgerungen.....	88
5.6 Implementierung der Views.....	89
5.6.1 PureMVC Views & Mediators.....	90
5.6.1.1 View Basisklassen.....	90
5.6.1.2 Textfeld.....	92
5.6.1.3 Eingabefeld.....	93
5.6.1.4 Schaltfläche.....	95
5.6.1.5 View & Mediator Zusammenspiel.....	96
5.6.1.6 Erstellte Views & Mediatoren.....	99
5.6.2 SproutCore Views.....	102
5.6.2.1 Hauptoberfläche.....	102
5.6.2.2 Contentbereich.....	108
5.6.2.3 Formular/Eingabemasken.....	110
5.6.2.4 Listen-Ansicht.....	112
5.6.3 Schlussfolgerungen.....	116
5.7 Implementierung der Controller.....	117
5.7.1 Flash/PureMVC Controller.....	117

5.7.2 SproutCore Controller.....	121
5.7.2.1 Routes.....	121
5.7.2.2 ObjectController.....	124
5.7.2.3 ArrayController.....	127
5.7.2.4 Erzeugte Controller.....	128
5.7.3 Schlussfolgerungen.....	134
5.8 Zusammenfassung und Fazit.....	136
6. Ausblick.....	139
6.1 Ist HTML5 ein Flash-Killer?.....	140
6.2 HTML5 und wie weiter?.....	140
Abkürzungsverzeichnis.....	142
Glossar.....	143
Abbildungsverzeichnis.....	144
Literaturverzeichnis.....	145
Selbstständigkeitserklärung.....	147

## **1 Einleitung**

Neben der Erfindung der Telekommunikation und des Fernsehens gibt es wahrscheinlich keine Erfindung, die die Gesellschaft der Industrie- und Schwellenländer entscheidender prägt als das Internet. Einer der bekanntesten Dienste des Internets, das World Wide Web kurz WWW, erlebt einen stetigen Anstieg an Nutzerzahlen. Soziale Netzwerke ziehen immer mehr und vor allem jüngere Nutzergruppen in ihren Bann und verändern somit die Kommunikation von Generationen.

Ein ganzer Industriezweig beschäftigt sich damit dem Verbraucher das Leben zu vereinfachen in dem sämtliche Informationen und Einkaufsmöglichkeiten als Dienstleistung im Internet verfügbar gemacht werden. Der Trend führt sogar soweit, dass Menschen ihre Daten nicht mehr auf externen Datenträgern transportieren, sondern diese ganz einfach auf Webspaces ablegen und somit unter der Voraussetzung eines Internetzugangs der Zugriff von überall ermöglicht wird. Onlinebanking, Videokonferenz/Videochat, soziale Netzwerke, all das sind Begriffe die in den letzten Jahren an Bedeutung gewonnen haben und die es nicht geben würde, wenn sich das Internet nicht seit Anfang an stetig weiterentwickelt hätte. Wohin diese Entwicklung führen wird, ist schwer zu sagen. Fest steht jedoch, dass das Internet einen wesentlichen Teil im Leben aktueller und zukünftiger Generationen einnimmt und einnehmen wird.

Seit seiner Gründung am 01. Dezember 1994 bemüht sich das World Wide Web Consortium (W3C) einen einheitlichen Standard zu erarbeiten und die nötigen Rahmenbedingungen festzulegen. Webentwickler weltweit ersehnen einen solchen seit Jahren, würde er doch unnötigen Programmieraufwand für verschiedene Browsertypen und -versionen eliminieren. Bis dieser Standard vollständig erarbeitet und etabliert ist,

wird es aber noch einige Jahre dauern. Die Problematik liegt darin, dass er für die Zukunft gewappnet und die Bereitstellung älterer Webseiten ebenso kompromisslos zu bewerkstelligen sein muss. Dieser doppelte Anspruch verzögert die Fertigstellung ungemein.

## **1.1 Motivation**

Das Internet ist der weltweit größte Netzverbund und eine Zusammenfassung von verschiedenen Diensten. Das WWW – es stellt den am häufigsten genutzten Dienst im Internet dar – hat einen sehr großen Anteil an der in den 90er Jahren entstandenen und stetig wachsenden Beliebtheit des Internets. Es hat sich in dieser Zeit vom reinen Wissenschaftsnetz zum kommerziell genutzten Netz, einem Super-Informations-Highway mit einer Vielzahl von Diensten und multimedialen Anwendungen, gewandelt. Aus dem Leben vieler Bürger in Industriestaaten ist das Internet fast nicht mehr wegzudenken. Es ermöglicht eine flexiblere Gestaltung des Privatlebens durch Zeitersparnis, Kontaktpflege mit Familie und Bekannten, Arbeitserleichterungen und vielem mehr. Die Möglichkeiten, die das heutige Internet bietet, sind fast grenzenlos. Soziale Plattformen, Communities genannt, wie z.B. Facebook oder StudiVz bilden für eine Großzahl von Menschen die Grundlage ihrer sozialen Interaktion und dementsprechend hoch sind die Ansprüche der Nutzer an selbige.

Aus diesem Grund muss heutzutage sehr gut abgewägt werden, wie ein Projekt umgesetzt werden soll. Die zu nutzende Technologie spielt zum einen aus Sicht der möglichen Nutzer und deren Systemvoraussetzungen (CPU-Leistung, Browser-Hersteller und -Version) eine entscheidende Rolle und zum anderen stellen Entwicklungs- und Wartungsaufwand eine nicht zu unterschätzende Größe dar.

## **1.2 Zielsetzung**

Diese Diplomarbeit soll zwei Technologien vergleichen, die auf den ersten Blick in Konkurrenzkampf stehen, bei genauer Betrachtung jedoch nicht wirklich als Konkurrenten bezeichnet werden können.

Flash mit seiner Browserunabhängigkeit bietet einem Entwickler die Möglichkeit mit einmaligem Entwicklungsaufwand viele Nutzer zu erreichen. Voraussetzung ist ein installiertes Flash-Player-Plugin auf dem Computer des Seitenaufrufers. Dieses Plugin ist der von der Internetgemeinde als großes Manko angesehene Stolperstein. Wieso dem Nutzer die Installation eines extra Programms auferlegen, wenn man ähnliche Funktionalität ohne diesen Zwischenschritt anbieten kann? HTML und die verschiedensten Technologien, die nötig sind um annähernd vergleichbare Ergebnisse zu liefern – wie es mit Flash der Fall ist – verlangen einem Entwickler die Kenntnis mehrerer Programmiersprachen und deren Verknüpfung ab. Dieser Umstand in Kombination mit verschiedensten Browsern und deren Versionen macht es zu einer schwierigen Aufgabe ein Produkt zu entwickeln, dass in allen Browsern dieselbe Funktionalität zur Verfügung stellt.

HTML5 steht als Weiterentwicklung der bestehenden HTML 4.01 Spezifikation in den Startlöchern das WWW zukunftssicher zu machen. Es soll besagten Aufwand minimieren und gleichzeitig Möglichkeiten bieten, die Flash schon seit geraumer Zeit gewährleistet. Deshalb soll anhand des prototypischen Entwurfs einer Plattform zum Tauschen von Konsumgegenständen evaluiert werden, welche der zu vergleichenden Technologien ein Webentwickler einsetzen sollte, um optimal für die Zukunft aufgestellt zu sein.

## 2 Begriffserklärungen

Bevor in den nächsten Kapiteln die zwei Technologien Flash und HTML5 vorgestellt und im Anschluss anhand einer prototypischen Anwendung verglichen werden, wird in diesem Kapitel zunächst die Geschichte des Internets vorgestellt. Beide Technologien nehmen einen großen Stellenwert in den Diskussionen um dieses Medium ein, wodurch diese Informationen grundlegend erforderlich sind. Im Anschluss werden die Arbeitsgruppen die sich mit HTML5 beschäftigen und Begriffe der Thematik Web2.0 erläutert.

### 2.1 Die Geschichte des Internets

Der Name Internet kommt von dem englischen Begriff „Interconnected set of networks“, was soviel bedeutet wie „miteinander verbundene Netzwerke“. Das Internet ist heute ein weltumspannendes Netz von vielen einzelnen Computernetzwerken.

Durch die Erfindung des elektrischen Stroms im Jahre 1730 war der Grundstein für die elektrische Telegrafie gelegt, welche Samuel Thomas von Soemmerring im Jahr 1809 erfand.

Die Erfindung der elektromagnetischen Induktion im Jahr 1832 durch Michael Faraday führte zu den ersten Versuchen mit einem elektromagnetischen Telegrafen, wodurch 1833 die erste Telegrafische Nachrichtenübertragung gelang.

Ab diesem Zeitpunkt befassten sich mehrere Personen mit der Erforschung und Entwicklung von Apparaturen zur Übertragung von akustischen Signalen.

Schlussendlich war es Alexander Graham Bell – im Jahr 1876 –, der die Fertigkeiten

## 2 Begriffserklärungen

---

zusammen brachte, das Telefon über eine Versuchsapparatur hinaus als Gesamtsystem zur Marktreife zu bringen.

Die Datenübertragung mittels Impulsen, die durch Kabel übertragen werden, stellt bis heute das wichtigste Transportmittel von Informationen dar.

Aufgeschreckt durch *Sputnik*, dem ersten von der UdSSR ins All beförderten geostationären Satellit, gründete das Verteidigungsministerium der USA die „Advanced Research Projects Agency“. Die ARPA hatte die Aufgabe neue Technologien im Bereich der Datenübertragung und Kommunikation zu entwickeln.

Das führte Ende der 1950er Jahre, durch J. C. R. Licklider und seinem Forschungsteam, zur Entwicklung des ersten „Time-Sharing-Systems“ der Welt. Dabei war ein Zentralrechner mit sternförmig angeschlossenen Terminals verbunden, wodurch mehrere Nutzer gleichzeitig dessen Rechenleistung nutzen konnten. Damit wurde es ermöglicht große geografische Distanzen mit Terminals eines Herstellers zu überwinden, allerdings war die Anzahl der Anschlüsse begrenzt. Für militärische Zwecke eignete sich dieser Netzaufbau jedoch nicht, weil eine Störung des Zentralrechners den Ausfall des gesamten Netzes bedeutete. Ein weiterführender Ansatz war das dezentrale Netzwerk, mit dem sich Paul Baran beschäftigte. Hierbei wurden mehrere Zentralrechner, die jeweils über die sternförmig angeschlossenen Terminals verfügten, miteinander verbunden. Fiel ein Zentralrechner aus, war das Netzwerk im Ganzen zwar geschwächt, aber immer noch in Teilen nutzbar. Um bei einem Netzwerkschaden so wenig wie möglich Terminals zu verlieren, entwickelte Paul Baran das „Distributed Network“ bei dem der Zentralrechner überflüssig wurde. Jedes Terminal, das mit dem Netzwerk verbunden war, hat alle Funktionen an Board, die für die Kommunikation im Netzwerk notwendig sind und galt somit als Computer. Ein weiterer Vorteil mehrerer Verbindungen zu einem möglichen Zielcomputer führte zur Entwicklung des „Packet-Switching“, wobei eine Datei nicht mehr als Ganzes

## *2 Begriffserklärungen*

---

übertragen, sondern in viele Datenpakete gleicher Größe zerteilt und einzeln transportiert wird. Staus werden somit vermieden, denn pro Paket kann die Route neu berechnet werden und muss bei einer fehlerhaften Übertragung nicht noch einmal als Ganzes übertragen werden. Diese intelligente Übertragung beschleunigte und entlastete das Netzwerk wesentlich. 1965 wurde diese Art der Netzkommunikation zum ersten Mal eingesetzt. 1966 begann die Planung zur Vernetzung aller über das Land verteilten Computerzentren der ARPA nach Paul Barans „Distributed Network“. Gegen Ende 1969 wurden nacheinander die großen Computerzentren verbunden.

Die angestrebte Unabhängigkeit von Herstellern und Betriebssystemen führte zur Entwicklung des „Interface Message Processor“, kurz IMP, durch Wesley Clark. Der IMP ist ein Minicomputer mit einem einheitlichen Netzwerkprogramm, der an jeden Großrechner angeschlossen wurde und für die Datenübermittlung zuständig war. Durch diese Verbindung der großen Computerzentren der ARPA entstand das ARPANET und zum ersten Mal waren Rechner verschiedenster Art miteinander verbunden.

Die ersten Anwendungen im ARPANET waren „Telnet“ und „FTP“ bevor Ray Tomlinson 1972 eine Software zum Verschicken und Empfangen elektronischer Post veröffentlichte. Der E-Mail-Dienst war geboren und führte zu einem sprunghaften Anstieg der Nutzerzahlen. Weitere Verbreitung erfuhr das ARPANET durch Vorführung auf der International Conference on Computer Communications und Weitergabe des Wissens unter anderem an die NASA, die Air Force und Universitäten. Darauf folgte das ALOHANET, das Forschungsstationen auf Hawaii vernetzte und sich aufgrund störungsanfälliger Telefonleitungen zum PRNET (Packet Radio Network) weiterentwickelte, in dem Daten per Radiowellen übertragen wurden. Als weitere Kommunikationsmöglichkeit wurde 1973 das SATNET (Satelliten Network) entwickelt. Die Nachteile der verschiedenen Datenübertragungsarten führten schließlich 1974 Vinton Cerf und Bob Kahn zur Entwicklung eines umfassenden Netzwerkprotokolls,

## *2 Begriffserklärungen*

---

dass sich auf ein einheitliches Datenformat und eine einheitliche Verbindungsmethode beschränkte, dem Transmission Control Protocol / Internet Protocol, kurz TCP/IP. 1983 wurde das TCP/IP zum Standard erklärt und kommt noch heute in der Form zum Einsatz. Im selben Jahr entstand das MILNET (Military Network) in das der komplette militärische Bereich des ARPANET verlagert wurde. Das ARPANET diente ab dem Zeitpunkt nur noch rein zivilen Gruppen. Dieser Schritt war notwendig geworden, weil sich immer mehr internationale Netzwerke anschlossen und somit die Unabhängigkeit des Militärs in Gefahr geriet. Bis 1985 wurden die bestehenden Netzwerke vorrangig von kleineren und größeren Nutzergruppen zur Kommunikation und zum Austausch ihrer akademischen Forschungsthemen genutzt.

1985 gab es eine entscheidende Änderung. Die Netzwerkzugänge des 1984 gegründeten JANET (Joint Academic Network) und des NSFNET (National Science Foundation Network) wurden für alle Benutzer – egal aus welchem Forschungsbereich – geöffnet. Dadurch erlangte das NSFNET immer mehr an Beliebtheit und übernahm schließlich 1990 die Funktion des ARPANET. In diesem Schritt wurde das ARPANET eingestellt. Im selben Jahr schlossen sich die Netzwerke von Canada, Dänemark, Finnland, Frankreich, Island, Norwegen und Schweden an das NSFNET an. 1991 folgten dann Deutschland, Japan, Niederlande und das Vereinigte Königreich.

Der Dienst der in der heutigen Zeit von vielen Menschen mit dem Internet gleichgesetzt wird ist das World Wide Web. Es ist 1990 aus einer Weiterentwicklung eines etwas älteren Projekts von Tim Berners-Lee – er war in den 80er und 90er Jahren Informatiker am Hochenergieforschungszentrum CERN – hervorgegangen und führte zu einer rasant ansteigenden Beliebtheit des Internet.

## **2.2 Browser**

Ein Browser ist ein spezielles Programm, das zur Darstellung von Inhalten aus dem Internet (World Wide Web) genutzt werden kann. Er übersetzt die im Quellcode verwendeten Befehle in das entsprechende Aussehen. Browser sind die Benutzeroberfläche für Webanwendungen, aber auch andere Daten und Dokumente können betrachtet werden. Es gibt mehrere Hersteller, die die Konventionen des W3C (siehe 2.3) unterschiedlich umsetzen. Diese auseinander gehenden Herangehensweisen führen dazu, dass Webseiten in verschiedenen Browsern unterschiedlich dargestellt werden. In unterschiedlichen Versionen eines Browsers können Konventionen unterschiedlich umgesetzt worden sein, was die versionsübergreifende Entwicklung für Programmierer zu einer enormen Herausforderung macht. Zu den Merkmalen eines guten Browsers zählen Schnelligkeit, Leistungsbedarf auf dem Rechner des Anwenders und die Umsetzung der neusten Innovationen im World Wide Web.

## **2.3 W3C**

Das W3C ist ein internationales Konsortium, in dem Mitgliedsorganisationen, ein fest angestelltes Team und die Öffentlichkeit gemeinsam daran arbeiten, Web-Standards und Richtlinien zu entwickeln – daher der Name World Wide Web Consortium, kurz W3C. Gegründet wurde es am 1. Oktober 1994 von Tim Berners-Lee, dem Erfinder des World Wide Web. Die Entstehung des W3C ist eng mit der Entstehung des World Wide Web verbunden. Die große Gefahr von Inkonsistenzen in der Nutzung vorliegender Technologien könnte zu unwirksamen Verknüpfungen führen und das galt es möglichst zu verhindern. Die Grundherangehensweise an neue Technologien oder Teilaspekte

## *2 Begriffserklärungen*

---

besteht darin, den kleinsten gemeinsamen Nenner zu finden und diesen zu einer Spezifikation zu verarbeiten, so dass diese von allen Mitgliedsorganisationen unterstützt wird.

### **2.4 WHATWG**

Die Web Hypertext Application Working Group, kurz WHATWEG, ist eine Arbeitsgruppe, die von mehreren Unternehmen, darunter unter anderem Mozilla Foundation, Opera Software ASA und Apple Inc., betrieben wird und deren Ziel die Entwicklung neuer Technologien zur einfacheren Erstellung von Internetanwendungen ist. Die WHATWG wurde 2004 gegründet, weil aus Sicht der Browser-Hersteller das W3C die Entwicklung von HTML – zu Gunsten von XHTML – vernachlässigte und auf die Bedürfnisse von Programmierern zu wenig einging. Als Unterschied zu den Bestrebungen der W3C muss festgehalten werden, dass die WHATWG keinen festen Standard als Zielsetzung verfolgt. Vielmehr soll HTML weiterentwickelt werden, ohne aber auf einen definierten Stand hin zu arbeiten. Anders das W3C, dessen aktuelle Bestrebungen ganz klar auf HTML5 ausgelegt sind.

### **2.5 Plattformunabhängigkeit**

Plattformunabhängigkeit bedeutet bei Software, dass sie ohne extra Aufwand und ohne extra Hilfssoftware auf unterschiedlichen Systemen ausgeführt werden kann. Genauer betrachtet ist darunter die Eigenschaft eines Programms zu verstehen, auf unterschiedlichen Computersystemen mit Unterschieden in Architektur, Prozessor,

Compiler, Betriebssystem und weiteren Dienstprogrammen, die zur Übersetzung oder Ausführung notwendig sind, lauffähig zu sein.

### 2.6 Rich Internet Application

Der Begriff „Rich Internet Application“ kurz RIA (reichhaltige Internetanwendung) wurde von Macromedia im Jahre 2002 erstmalig verwendet. Damalige Internetangebote entwickelten sich zunehmend vom reinen statischen Wiedergeben von Inhalten zu vom Nutzer manipulierbaren dynamischen Inhalten, wodurch der Begriff *Web2.0*<sup>1</sup> entstand. Der Begriff RIA bezeichnet ein Konzept und basiert meist auf mehreren Web-Technologien wie zum Beispiel HTML, CSS, JavaScript und AJAX oder Flash und PHP. Um als RIA zu gelten, muss eine Anwendung über das Internet verfügbar sein. Allerdings muss sie nicht zwangsläufig im Browser ausgeführt werden, sondern kann auch als Desktopanwendung zum Einsatz kommen. RIAs im heutigen Sinn bieten dem Anwender Möglichkeiten, die früher Desktopanwendungen zur Verfügung stellten. Die Individualisierung und Vereinfachung der Internetnutzung spielt eine entscheidende Rolle bei der Entwicklung neuer Anwendungen. Unter Interaktion versteht man beispielsweise das Ändern der Gestaltung, das Erzeugen und Verwalten von Daten, Drag&Drop-Funktionalität, sowie die Verwendung von Tastenkürzeln.

---

<sup>1</sup> [http://de.wikipedia.org/wiki/Web\\_2.0](http://de.wikipedia.org/wiki/Web_2.0)

## **3 Flash**

In diesem Kapitel wird zunächst anhand der Geschichte von Flash die Entstehung und Veröffentlichungen bis zum aktuellen Stand beschreiben. Im Anschluss wird die Programmiersprache ActionScript, spezifische Dateiformate, Einsatzbereiche und häufig verwendete Techniken vorgestellt. Anschließend werden Vor- und Nachteile genannt und ein Fazit abgegeben.

### **3.1 Die Geschichte von Flash**

Als Jonathan Gay und Charlie Jackson – der Gründer von Silicon Beach Software – 1993 FutureWave Software gründeten, war ihnen sicher nicht bewusst, mit dieser Firma und dem 1994 erschienenen vektorbasierten Zeichenprogramm SmartSketch den Grundstein für das einmal am meisten verbreitete Freeware Programm der Welt gelegt zu haben.

SmartSketch wurde aus der Intention heraus entwickelt, den Arbeitsprozess für Gestalter zu vereinfachen, indem die Möglichkeit geschaffen wurde, mittels Grafiktablets Zeichnungen – ähnlich dem Zeichnen auf Papier – zu erstellen. Jonathan Gay wollte eine Software entwickeln, die es dem Anwender erlaubt, artgemäß dem Erstellen von Lego-Objekten Schritt für Schritt im Gestaltungsprozess Fortschritte zu machen. Dieses Schachtelungsprinzip findet sich noch heute in Flash wieder. Im Sommer 1995 erhielt FutureWave zunehmend positive Resonanz zu SmartSketch sowie Vorschläge das Grafikprogramm zu einem Animationsprogramm weiterzuentwickeln. Zur selben Zeit trat ein neues Medium in das Auge der Öffentlichkeit, das Internet. Gay und seine Kollegen sahen darin das Potential, dass ein Großteil der zukünftigen

### 3 *Flash*

---

Internetnutzer reges Interesse daran haben könnte, Grafiken und Animationen anzubieten bzw. auszutauschen.

Daraufhin erweiterte Gay SmartSketch kurzerhand um Animationsmöglichkeiten. Mittels Java wurde ein Browserplugin entwickelt, das die erstellten Animationen, die im Splash Format (SPL) vorlagen, im Browser abspielen konnte.

Im Herbst 1995 erschien der Netscape Browser mit dem 'advanced programming interface', welche es erlaubte den Browser zu erweitern. Auf Grund des Wiedererkennungswertes wurde SmartSketch erst in CelAnimator und später in FutureSplash Animator umbenannt.

Der größte Erfolg stellte sich im August 1996 ein, als Microsoft für seine MSN Web-Version ein fernsehähnliches Erlebnis schuf und das zur Umsetzung eingesetzte Programm FutureSplash Animator war. Ein zweiter großer Kunde war Disney Online, welcher FutureSplash nutzte, um Animationen und Schnittstellen für den Bezahlendienst „Disneys Daily Blast“ zu erstellen. Dadurch, dass Disney zum selben Zeitpunkt den Macromedia Shockwave Player nutzte, kam über diesen Umweg der Kontakt mit Macromedia zustande. Im Dezember 1996 kaufte Macromedia FutureWave und aus FutureSplash Animator wurde Macromedia Flash 1.0. Im Zuge der Produktumbenennung wurde aus dem Abspielformat Splash das SWF.

Seit dem Verkauf an Macromedia erschienen im Jahresrhythmus neue Versionen von Flash. Nach und nach stieg der Funktionsumfang und dem Gestalter wurden mehr Möglichkeiten zur Auslebung seiner Kreativität gegeben. Im Jahr 1999 erschien Version 4, die zum ersten Mal die so genannte Programmiersprache ActionScript integrierte.

Es konnten Variablen, bedingte Anweisungen, sowie Schleifen zur Programmablaufsteuerung eingesetzt werden. Dadurch war der Weg geebnet für mehr als nur reine Animationsentwicklung. Eingabetextfelder ermöglichten es komplexe Formulare zu erstellen und die eingegebenen Daten mittels dynamischer Webseiten zu verarbeiten. Im

### 3 *Flash*

---

Jahr 2000 erschien Version 5 in der ActionScript stark verändert und an den ECMAScript-Standard angepasst wurde. JavaScript-Entwicklern sollte dadurch ein schnellerer Einstieg in Flash ermöglicht werden, da JavaScript auf diesem Standard aufbaute. Eine weitere Neuerung war der Debugger, der den Entwicklungsprozess und die damit verbunden häufige Fehlersuche in Flash-Projekten stark vereinfachte. 2002 erschien Flash MX (Version 6) mit entscheidenden Erweiterungen wie der Zeichnen-API, die es erlaubte dynamische Formen zu erstellen. Zusätzlich war ein Videocodec integriert und die Unterstützung für Unicode gegeben. Im Oktober 2003 kam Flash MX 2004 auf den Markt und damit ActionScript 2.0.

Im Juni 2004 erscheint die aktualisierte Version Flash MX 2004 7.2 in der Stabilitätsprobleme behoben und die Leistung stark verbessert wurde. Mitte Juni stellt Macromedia die „Flash Plattform“ vor, die in erster Linie an Unternehmenskunden adressiert war. Am 8. August wurde zusammen mit „Studio 8“ Flash Professional 8 vorgestellt. Flash 8 enthielt etliche Neuerungen, unter anderem wurde die Möglichkeit geschaffen Rastergrafiken zu erstellen und zu manipulieren. Filter wie zum Beispiel Gaußscher Weichzeichner, Schlagschatten oder Verzerrungen, Datei-Upload und eine neue Text-Engine namens FlashType, Bitmap-Caching, einstellbares Easing, ein neuer Videocodec mit Alphakanal-Unterstützung (Transparenz), ein stand-alone Video-Encoder mit Stapelverarbeitung, sowie eine verbesserte Programmieroberfläche sind hier als die bedeutendsten zu nennen. Im selben Jahr übernimmt Adobe Macromedia für 3,4 Milliarden US-Dollar. Adobe führt die Bezeichnung *Macromedia Flash* bis zum nächsten Produktzyklus fort, danach heißt das Produkt *Adobe Flash*.

Im Juni 2006 erscheint der Adobe Flash Player 9 für Windows und Mac OS X mit speziellen Anpassungen für eine bessere Integration in Adobe Flex 2. Eine weitere Innovation stellt die Integration von ActionScript 3.0 dar. Des Weiteren ist ein JustInTime-Compiler Bestandteil, welcher mit anderen Verbesserungen einen großen

### *3 Flash*

---

Geschwindigkeitsvorteil bei der Scriptausführung gegenüber den älteren ActionScript-Versionen mit sich bringt. Seit Mai 2007 ist Adobe Flash CS3 verfügbar, diese Version ermöglicht direkte Importe aus Adobe Photoshop und Illustrator, was zu einer enormen Erleichterung und Beschleunigung des Arbeitsprozesses führt. Im Jahr 2008 legt Adobe die Spezifikation für Flash offen, was die komplette Indizierung durch Suchmaschinen ermöglicht und somit ein großes Manko an SWF-Dateien beheben soll. Mitte Oktober erscheint Adobe Flash CS4 und der Adobe Flash Player 10.

Am 7. Mai 2010 erscheint Adobe Flash CS5, das diverse Neuigkeiten zur Prozessoptimierung bietet. Der Flash Builder ist nahtlos integriert und die Code-Hinweise, die so genannte IntelliSense, wurden stark verbessert. Ein weiteres neues Feature sind die Code-Snippets, die häufig verwendete Code-Bausteine zur Verfügung stellen. Videos können direkt in der Entwicklungsumgebung abgespielt werden, was der Interaktion mit Videos sehr zu Gute kommt. Ein sehr hilfreiches Werkzeug ist die neue Text-Engine. Diese erlaubt eine feinere Steuerung von Textelementen. Textfelder können per Mausklick in Spalten eingeteilt und so miteinander verknüpft werden, dass Text dabei ab einem bestimmten Punkt in ein anderes Textfeld überläuft. Ein weiteres neues Feature ist die Unterstützung von XFL-Dateien, diese trennt Code, Layout und Assets voneinander. Dadurch wird der Austausch von Daten, z.B. Bildern, sehr vereinfacht.

Am 10. Juni 2010 erschien der Adobe Flash Player 10.1, der im wesentlichen Umfang Stabilität, Performance und Ressourcennutzung optimierte, um somit auch auf dem mobilen Endgerätemarkt bestehen zu können. Grundlegende 3D-Manipulationen und eine 3D-Zeichen-API sind integriert worden. Eine große Weiterentwicklung stellt das Auslagern von Prozessen auf die Grafikkarte dar. Somit wird die CPU entlastet und die Wiedergabe von H.264 Videos (HD) beschleunigt. Des Weiteren wurde die

Speicherverwaltung überarbeitet und Multitouch- sowie Lagesensoren-Unterstützung integriert. Letztere sind vorwiegend in Smartphones integriert und ermöglichen neue Möglichkeiten der Steuerung und Interaktion.

Mit der aktuellsten Flash Player-Version 10.3 stopft Adobe diverse kritische Sicherheitslücken, verbessert die Audioqualität und ermöglicht das Löschen von 'Local Storage Objects' über das Browsermenü.

## **3.2 ActionScript**

ActionScript, kurz AS, ist die Programmiersprache für die Adobe Player-Laufzeitumgebung. Sie ist eine objekt- und ereignisorientierte Programmiersprache, die speziell für die Webseitenanimation entwickelt wurde und in direktem Zusammenhang mit der Entstehung von Flash zu nennen ist. Der aktuellste Entwicklungsstand ist ActionScript 3.0 und stellt ein vielseitiges Werkzeug zur Umsetzung interaktiver und vielseitiger Projekte dar.

Der ActionScript-Programmcode wird in das sogenannte Bytecode-Format kompiliert und anschließend von der ActionScript Virtual Machine, kurz AVM, ausgeführt. Ein sogenannter Compiler übernimmt die Umwandlung in Bytecode und bettet diesen in eine SWF-Datei ein. Als Erfinder von ActionScript gilt Gary Grossman. Er verließ die Firma zeitig um sich anderen Projekten zu widmen.

### 3.2.1 ActionScript 1.0

ActionScript wurde erstmals in Flash 4 veröffentlicht und entsprach einer Sammlung von Aktionen die mittels „zusammenklicken“ in einen logischen Zusammenhang zu bringen waren. In dieser Version entspricht ActionScript einer Mischung aus Perl<sup>2</sup> und JavaScript. Für die Anwendung spezifische Werte konnten bereits in Variablen gespeichert und mittels Kontrollstrukturen gesteuert werden. In den Vorgängerversionen gab es auch Scripting Unterstützung, aber diese basierte auf keinem grundlegenden Standard und bezog sich nur auf einfache Zeitleistensteuerung. In der Flash 5 Version handelte es sich bei ActionScript erstmals um eine vollwertige Programmiersprache. Sie wurde an den ECMAScript 3 Standard angepasst, entsprach diesem aber nicht zu 100%. Zusätzlich wurden Events und Switch-Anweisungen eingeführt, wodurch erstmals gezielt auf Nutzeraktionen reagiert werden konnte. Eine weitere sehr nützliche Funktionalität waren *Prototypes*, die eine Vorstufe zu Klassen darstellen.

### 3.2.2 ActionScript 2.0

ActionScript 2.0 erschien im September 2003 zusammen mit dem Flash Player 7 und wurde mit Flash MX 2004 verfügbar. Die Entwicklergemeinde – sie war von Flash und ActionScript begeistert – teilte den Flash-Entwicklern Änderungs- und Erweiterungsvorschläge mit, wodurch ActionScript 2.0 für größere und komplexere Projekte ausgelegt werden konnte. Unter anderem wurden eine Laufzeitüberprüfung, klassenbasierte Syntax, die Schlüsselwörter *class* und *extends* eingeführt und die Anpassung an den ECMAScript 4 Standard vorgenommen. Als besonders gilt, dass

---

<sup>2</sup> [http://de.wikipedia.org/wiki/Perl\\_\(Programmiersprache\)](http://de.wikipedia.org/wiki/Perl_(Programmiersprache))

ActionScript 2.0 in ActionScript 1.0 Bytecode abgespeichert wird, damit es auch mit dem Flash Player 6 kompatibel ist, da dieser zu dem damaligen Zeitpunkt der am weitesten verbreitete war. Die Auslegung auf Objektorientierte Programmierung führte dazu, dass AS C++ und Java immer ähnlicher wurde. Die dadurch stark reduzierte Umstellungs- bzw. Einarbeitungszeit machte die Programmiersprache einer größeren Entwicklergemeinde zugänglich.

#### **3.2.3 ActionScript 3.0**

ActionScript 3.0 erschien im Juni 2006 zusammen mit dem Flash Player 9 und wurde mit Flash 9/CS3 verfügbar. Diese Version basiert auf der ECMAScript 4 Sprachspezifikation, wurde vollkommen überarbeitet und ist für vollständige objektorientierte Programmierung ausgelegt. AS 3.0 unterscheidet sich in seiner Architektur und Konzeption weitestgehend von seinen Vorgängern. In Folge dessen wurde die AVM neu konzipiert und für AS 3.0 in AVM2 umbenannt.

Als besondere Neuerung ist das Ereignismodell, das auf der DOM3-Ereignisspezifikation basiert, zu betrachten. Die verschiedenen visuellen Elemente einer AS 3.0 Anwendung sind in der Anzeigeliste – einer Art Baumstruktur – angeordnet. Jedes Element hat ein Vaterelement und eventuell Kinderelemente. Das Ereignismodell spielt in der Nutzung der neuen einheitlichen Klassen zum Laden von diversen Daten eine entscheidende Rolle. [Oreilly 2007]

Mit der URLLoader-Klasse lassen sich Text oder Binärdaten, z. B. XML-Dateien oder PHP-Skripte, laden. Um Zustandsänderungen des Ladevorgangs abfangen zu können, müssen relevante Ereignisse (Events) mit der Funktion '*addEventListener()*' direkt an das Objekt der URLLoader-Klasse „angehängt“ werden. Während des Download-

### 3 Flash

---

Vorgangs geben Benachrichtigungen (Events) über den Fortschritt Auskunft. Die zwei Eigenschaften *bytesLoaded* und *bytesTotal*, die als Eigenschaften des *ProgressEvent* abrufbar sind, bieten die Möglichkeit den Ladevorgang prozentual darzustellen. Ist der Ladevorgang abgeschlossen, sind die geladenen Daten über die Eigenschaft *data* des Events abrufbar.

Zum Laden von SWF-Dateien und Grafiken (JPG, PNG, GIF) ist die *Loader*-Klasse vorgesehen, bei der die Event-Listener an die Eigenschaft *contentLoaderInfo*, ein Objekt das dem zu ladenden Datenobjekt entspricht, angehängt werden.

Die neue *URLStream*-Klasse<sup>3</sup> stellt noch während des Downloadvorgangs Zugriff auf die Daten als unformatierte Binärdaten bereit, wodurch z. B. Informationen wie der Dateityp und die Auflösung eines Bildes ermittelt werden können.

AS3.0 unterstützt Pakete und Namespaces<sup>4</sup>, wodurch besonders die Strukturierung von Projekten verbessert und Namenskonflikte vermieden werden.

Eine weitere sehr nützliche Neuerung stellt die XML-API dar, die auf der E4X-Spezifikation beruht [Oreilly 2007]. XML-Daten werden nun als nativer Datentyp behandelt, dies führt zu einer enormen Steigerung der Verarbeitungsgeschwindigkeit seitens der AVM und zusätzlich zu einer einfacheren Verarbeitung und Nutzung von XML-Daten.

Die Aufnahme der numerischen Grunddatentypen „int“ als z.B. Schleifenzähler-Variable und „uint“ als z.B. Farbwert stellt im Verhältnis zu AS 2.0 nicht nur für Programmierer ein nützliches Mittel dar. Die Vorteile der schnellen Ganzzahlarithmetik der CPU können damit genutzt werden und führen ihrerseits zu einer kürzeren Verarbeitungszeit. In AS 3.0 wird eine strikte Typisierung vorausgesetzt und die vom Programmierer gesetzten Datentypen bleiben zur Laufzeit erhalten. Während des

<sup>3</sup> [http://livedocs.adobe.com/flash/9.0\\_de/ActionScriptLangRefV3/flash/net/URLStream.html](http://livedocs.adobe.com/flash/9.0_de/ActionScriptLangRefV3/flash/net/URLStream.html)

<sup>4</sup> [http://help.adobe.com/de\\_DE/ActionScript/3.0\\_ProgrammingAS3/WS5b3ccc516d44bf351e63e3d118a9b90204-7f9e.html](http://help.adobe.com/de_DE/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d44bf351e63e3d118a9b90204-7f9e.html)

Kompilierens und zur Laufzeit wird eine Datentypüberprüfung durchgeführt, wodurch die Datentypsicherheit des Systems stark verbessert wird.

Alle textbezogenen Schnittstellen wurden im *flash.text* Paket zusammengefasst. Zusätzlich wurde die *TextLineMetrics*-Klasse hinzugefügt, die sehr nützliche Informationen über eine einzelne Zeile oder ein Zeichen liefert.

Ab Flash CS5 und somit Flash Player Version 10 wurde die Text-Engine durch das neue TLF-Text Format erweitert, wodurch Funktionalitäten geboten werden, die Flash-Programmierern enorm viel Arbeit abnehmen. Es lassen sich nahezu alle Text-Manipulationen durchführen, die in Layout-Werkzeugen schon lange möglich sind. Zum Beispiel sind mehrspaltige Textfelder, Textfluss um Objekte und Textfeld-Verknüpfungen, die „überfließenden“ Text ermöglichen, realisierbar.

Die neue Sound-API bietet mittels *SoundChannel*- und *SoundMixer*-Klasse umfangreiche Funktionen. Audio lässt sich nun synthetisieren, manipulieren und abspielen. Darüber hinaus ist eine exakte Steuerung einzelner und aller in einer Anwendung befindlicher Sounds möglich.

Im Bereich Video wird ab dem Flash Player 10.1 die Hardware beschleunigte Dekodierung von H.264-Videos eingeführt. Erfüllt der Client die notwendigen Voraussetzungen, wird die Dekodierung auf die Hardware des Clients ausgelagert. Somit wird die CPU weniger beansprucht, was zu einer besseren Performance des Flash Players und einer niedrigeren Systemauslastung führt. Insbesondere auf mobilen Geräten wird dadurch der Akku geschont.

## **3.3 Dateiformate**

### **3.3.1 .fla**

Eine FLA Datei ist eine unkomprimierte Projektdatei und enthält alle Ausgangsmaterialien und Animationen. Mit dem entsprechenden Flash Autoringwerkzeug kann sie bearbeitet und zu einer kompiliert (veröffentlicht) werden.

### **3.3.2 .swf**

Eine SWF-Datei ist ein komprimiertes Flashprojekt (Bytecode) und kann von einem Flash Player abgespielt werden. Bei der Bedeutung, was SWF ausgeschrieben heißt, gibt es keine eindeutige Festlegung. Es gibt zwei Varianten, die in etwa gleich viel Bedeutung finden. Zum einen *ShockWaveFlash*, sie entspricht der Bezeichnung, die Macromedia als 3D-Format der Director-Umgebung vermarktet hatte. Die andere Variante ist *SmalWebFormat*, was den Eigenschaften des Formates, geringe Dateigröße und enorme Skalierbarkeit ohne Qualitätsverlust, am ehesten entspricht. Die geringe Dateigröße und Skalierbarkeit resultieren daraus, dass Grafiken, die in Flash erstellt wurden, als Vektorgrafik abgespeichert werden.

### **3.3.3 .as**

Eine ActionScript-Datei enthält ausschließlich ActionScript Anweisungen und kann mit jedem Texteditor bearbeitet werden. Zum Beispiel kann jedes Objekt in einem Flash-Projekt als Klasse in einer .as-Datei gespeichert werden. Dieses Format ermöglicht eine bessere Strukturierung von Projekten, indem Design und Quellcode getrennt gespeichert werden.

### **3.3.4 .mxml**

MXML ist eine auf XML basierende Beschreibungssprache die von Adobe Flex und dem Adobe Flash Builder verwendet wird, insbesondere lassen sich mit ihr Komponenten beschreiben und mit Funktionalität versehen. Eingeführt wurde MXML, um die Entwicklung von RIAs und den Einstieg für Softwareentwickler zu vereinfachen.

### 3.3.5 Weitere Formate

- .swd - Eine vom Flash Debugger erzeugte Datei die Debug-Informationen enthält. Sie wird benötigt um Remote-Debugging durchzuführen und ermöglicht Einblick in die Struktur einer SWF
- .asc - Eine Datei die Server-seitiges ActionScript enthält
- .flv - Flash Video Format, eine Containerformat das ein Flashvideo enthält
- .swc - Eine ZIP-Datei die mit dem Flash-Authoring-Tool erstellt wird und Informationen über eine oder mehrere Flash-Komponenten enthält
- .jsfl - Eine Datei die Anweisungen zur Erweiterung des Flash-Authoring-Tools enthält
- .flp - Eine XML-Datei die alle Informationen über ein Flashprojekt enthält.
- .aso - Eine Datei die vom Flash Player erzeugt wird, um Local Shared Objects zu speichern.

## **3.4 Einsatzbereiche**

Durch die Integration der Programmiersprache ActionScript entwickelte sich Flash schnell von einer Gestaltungssoftware zu einem beliebten Alleskönner. Es wurde in den darauf folgenden Jahren alles entwickelt, was irgendwie technisch möglich war. Von einfachsten Werbebannern über Banner mit Nutzerinteraktionen und Synchronisierung mehrerer SWF-Dateien, hin zu kleinen Spielen, bis hin zu enorm umfangreichen Spielen mit integrierter Community, Produktkonfiguratoren, Umfrage-Werkzeuge, Präsentationen, Musik- und Video-Player und Desktop-Applikationen (AIR).

## **3.5 Techniken**

In diesem Kapitel werden Technologien vorgestellt, die in größeren Flash-Projekten – vornehmlich RIAs – zum Einsatz kommen. Darunter zählen unter anderem deeplinking, Suchmaschinenoptimierung und tracking. Die Einsatzgründe können aufgrund projektspezifischer Faktoren sehr vielfältig ausfallen. Relativ große Schwierigkeiten bereitet die Optimierung von Flash-Seiten für Suchmaschinenindexierung und die fehlende Unterstützung der Browser-History. Letztere steht stellvertretend für deeplinking und Favoriten-Links.

### **3.5.1 Deeplink**

Als ein relativ großes Manko von Flash-Seiten wird die schlechte Verlinkung von Unterseiten angesehen. In diesem Zusammenhang ist die fehlende Unterstützung der

Browser-History, die durch die Vor- und Zurück-Pfeile durchlaufen werden, zu erwähnen. Abhilfe schafft eine kleine *OpenSource* Bibliothek mit dem Namen *SWFAddress*<sup>5</sup>. Diese Bibliothek ermöglicht es, Flash-Seiten mit Deeplinking zu erstellen und setzt dabei auf die *ExternalInterface* Funktionalität. Dadurch besteht die Möglichkeit der Speicherung von Favoriten-Links und somit das direkte Aufrufen von Unterseiten. Des Weiteren ist mit *SWFAddress* zusätzlich die Interaktion mit dem Browser und dessen Vor- und Zurück-Pfeilen möglich, wodurch definierte Zustände der Flash-Seite durchlaufen werden können.

Eine andere Möglichkeit eine Deeplink-Unterstützung zu gewährleisten, wäre eine Hauptanwendung zu programmieren, die URL-Parameter (Content-ID oder ähnliches) auswertet und entsprechende Content-Seiten aufruft. Die Content-Seiten könnten in separate SWF-Dateien ausgelagert (Ladezeit minimieren) oder in die Hauptanwendung integriert werden.

#### **3.5.2 Suchmaschinen-Optimierung**

Ein großes Problem bei Flash-Seiten stellt die Optimierung für Suchmaschinen, kurz SEO, dar. Laut Adobe können die Googlebots seit Juli 2008 SWF-Dateien durchsuchen, aber nur nach statischem Text. Daran hat sich bis heute nicht viel verändert [adobe 1].

In großen Projekten ist es aus Gründen der Aktualisierbarkeit unüblich, Texte statisch anzulegen. Text wird vorrangig aus einer zuvor geladenen XML-Datei ausgelesen (siehe 3.5.3). Das Nachladen von Daten kann eine Suchmaschine nicht registrieren, wodurch die Daten nicht zur Indexierung verwendet werden können.

Um diese Daten dennoch der Suchmaschine zugänglich zu machen, bedarf es

---

<sup>5</sup> <http://www.asual.com/swfaddress/>

zusätzlicher Mechanismen. Der Content muss auf nicht Flash-Seiten zugänglich gemacht werden, wobei besonders acht auf die Umsetzung der SEO Richtlinien<sup>6</sup> gelegt werden sollte.

#### **3.5.3 Dynamisierung**

Die Inhalte von Flash-Seiten müssen nicht zwangsläufig statisch hinterlegt werden. Eine dynamische Lösung stellt das Auslagern von Texten und Grafiken dar. Bei dieser Vorgehensweise wird beim Start der Anwendung eine XML-Datei geladen, die alle nötigen Daten oder Verweise auf andere XML-Dateien enthält. Aus der XML werden Texte, die mit HTML-Formatierung versehen sein können, ausgelesen und angezeigt. Grafiken werden erst dann geladen, wenn diese benötigt werden.

#### **3.5.4 ExternalInterface**

Die ExternalInterface-Klasse stellt Funktionalität bereit, die es ermöglicht mit dem Container – vorrangig JavaScript in einer HTML-Seite – zu kommunizieren. Diese Klasse wurde als Erweiterung von FsCommand mit dem Flash Player 8 eingeführt. Die Kommunikation kann in beide Richtungen erfolgen. Als Voraussetzung muss der HTML Parameter *allowScriptAcces* auf *always* oder *sameDomain* gesetzt sein. Der dritte mögliche Wert ist *never* und blockiert die ExternalInterface-Funktionalität.

---

<sup>6</sup> <http://www.suchmaschinen-seo-marketing.de/home/suchmaschinenoptimierung-seo-richtlinien.html>

### 3.5.5 LocalConnection

*LocalConnection* findet Anwendung, wenn zwischen mehreren SWF-Dateien auf einem Client-System Daten ausgetauscht werden bzw. Einfluss aus einer SWF auf eine andere genommen wird. Die Verbindung kann auch zwischen unterschiedlichen Flash-Versionen hergestellt werden, da keine direkte Kommunikation der Programmiersprache erfolgt, sondern ausschließlich Daten übertragen werden. Leider ist diese Funktionalität nicht vollends ausgereift. Werden zum Beispiel SWF-Dateien von verschiedenen Servern geladen, ist die Gefahr hoch, dass keine Verbindung zu Stande kommt. Des Weiteren gibt es Probleme, wenn eine Web-Seite mit mehreren eingebundenen SWF-Dateien in mehr als einem Browser-Tab geöffnet wird. Da nicht mehr als ein Empfänger pro Verbindung unterstützt wird, funktioniert *LocalConnection* nur in der als erstes geladenen Seite.

Um diese Problematik zu beheben, kann eine externe Bibliothek namens *asLocalConnect*<sup>7</sup> verwendet werden, die von Markus Bordihn als eine Erweiterung von *LocalConnection* entwickelt wurde.

### 3.5.6 SharedObject

Ein *SharedObject* kann zur Speicherung von kleineren Daten – Logindaten oder Spielstände – auf dem Client-System genutzt werden. Diese Funktionalität wird seit dem Flash Player 6 unterstützt. Standardmäßig ist die zu speichernde Datenmenge auf 100kB beschränkt, kann aber vom Nutzer im Komponentenfenster (Rechts-Klick auf SWF-Datei) von 0 bis unbegrenzt verändert werden.

---

<sup>7</sup> <http://www.flashworker.de/tutorial/56/001.html>

### **3.6 Vor- und Nachteile**

Im Vergleich zu anderen Technologien - speziell gegenüber HTML/HTML5 - kann Flash mit zwei großen Vorteilen aufwarten. Zum einen ist die Plattformunabhängigkeit zu nennen. Eine SWF sieht auf jedem unterstütztem Betriebssystem und Browser – vorausgesetzt der nötige Flash Player ist installiert – identisch aus und kann auf gewohnte Weise benutzt werden. Dadurch ist theoretisch mit einmaligem Entwicklungsaufwand die größtmögliche Nutzergruppe erreichbar.

Da Adobe der alleinige Vertreiber der Flash-Technologie ist, kann relativ schnell auf Veränderungen am Markt reagiert und in relativ kurzen Zyklen – ein bis zwei Produktversionen – neue Funktionalitäten veröffentlicht werden, woraus eine überragende Flexibilität gegenüber HTML resultiert.

Den Vorteilen stehen genauso entscheidende Nachteile gegenüber. Suchmaschinen können z.B. SWF-Dateien nur nach statischem Text durchsuchen. Darauf wird in kommerziellen Projekten aufgrund der Aktualisierbarkeit weitestgehend verzichtet, wodurch sich für eine Suchmaschine eine leere Seite darstellt. In solchen Fällen muss viel Aufwand betrieben werden, um den Seiteninhalt dennoch zugänglich zu machen. Ein zweiter Nachteil ist das Flash Browser-PlugIn, der sogenannte Flash Player. Für Nutzer, die ihn nicht installiert haben, bleiben Flash-Inhalte verborgen. Nutzer, die ihn installiert haben, müssen ihn regelmäßig aktualisieren – ausgenommen Nutzer des Browsers Chrome –, um für Webseiten die den neusten Flash Player erfordern gerüstet zu sein.

### 3.7 Ausblick

Die Bemühungen sind groß, Flash aus seinen eingefleischten Bereichen zu verdrängen. Solange sich die Browser-Hersteller nicht auf einen gemeinsamen Codec einigen, wird es auf unabsehbare Zeit dabei bleiben, dass Flash den Bereich Video beherrschen wird. Zumal bis heute HTML5 die Techniken streaming, vor und zurück Spulen, Zeitanzeige, Kopierschutz, Vollbildmodus sowie Webcam nicht unterstützt und dadurch die Nutzerfreundlichkeit von HTML5 Videos nicht an die von Flash heran reicht. Darüber hinaus hat sich YouTube vorerst für Flash ausgesprochen, weil es bis heute am besten geeignet ist, um online Videos in vollem Umfang zugänglich zu machen und die Unterstützung mehrerer Formate eines Videos in keinem Verhältnis steht. [youtube 1]

Darüber hinaus wird das YouTube-Portal in den USA zur Streaming-Videothek<sup>8</sup>, US-Bürger können aktuelle Spielfilme als Stream mieten und anschauen, ausgebaut und dabei wird ausschließlich der Flash Player eingesetzt.

Die verbreitete Meinung, Flash verbraucht enorm viel Ressourcen des Clients, kann nicht ohne weiteres verallgemeinert werden. Es entscheiden drei Ebenen (Flash, Browser und Client-Architektur) darüber, wie sehr die Ressourcen des Systems in Anspruch genommen werden. Außer acht gelassen werden sollte auch nicht, dass das Dekodieren eines HD-Videos mit rechenintensiven Abläufen verbunden ist. Von Seitens Adobe hat sich diesbezüglich in letzter Zeit viel getan. Die Hardwarebeschleunigung des H.264 Codec hat die Ressourcenhungrigkeit von Flash enorm verringert. Daraus resultiert, dass Flash auch auf mobilen Endgeräten – Apple Produkte sind ausgenommen – weiterhin für die Wiedergabe von Videos zuständig sein wird.

In wieweit die Bemühungen von Adobe, die 3d-Technologie<sup>9</sup> besser in den Flash Player

---

<sup>8</sup> <http://www.golem.de/1105/83359.html>

<sup>9</sup> <http://labs.adobe.com/technologies/flashplatformruntimes/incubator/features/molehill.html>

### *3 Flash*

---

zu integrieren, dazu führen könnten, dass sich daraus eventuell eine neue Form des Internets entwickeln könnte, kann heute noch nicht abgeschätzt werden.

Die seit einigen Jahren verbreiteten Ankündigungen, HTML5 wird Flash aus dem Internet verdrängen, haben sich bis heute nicht bewahrheitet und es wird in geraumer Zeit auch nicht dazu kommen. Bei den geführten Diskussionen wird zumeist eines nicht beachtet, der Einsatzbereich von Flash und HTML5 ist keines Falls der Selbe, vielmehr ergänzen sich beide Technologien. Flash sollte nur dort eingesetzt werden, wo es sinnvoll ist. Eine Web-Seite mit reinem Text sollte möglichst nicht mit Flash erstellt werden. Schlussendlich ist entscheidend, was mit einer Webseite erreicht werden soll. Fragen, wie z. B. welche Zielgruppe angesprochen, welche Reaktionen hervorgerufen und was nach außen für eine Kernaussage erzielt werden soll, müssen vor Entwicklungsbeginn geklärt werden.

## 4 HTML5

Dieses Kapitel stellt im ersten Abschnitt die Geschichte von HTML vor, danach werden die Entwicklungsschritte der HTML-Spezifikation und anschließend die Neuerungen der sich in der Entwicklung befindlichen HTML5-Spezifikation vorgestellt.

### 4.1 Die Geschichte von HTML

HTML steht für Hyper Text Markup Language, was übersetzt soviel wie Hypertext-Auszeichnungssprache bedeutet.

Einer der Gründe für die Erfindung von HTML war, dass Wissenschaftler einen Weg suchten Informationen unkompliziert untereinander auszutauschen bzw. zugänglich zu machen. Ein großes Problem stellte sich schnell heraus, die Formatierung der Texte. Informationen wurden bis dato - Anfang der 90er Jahre - in einfachen ASCII-Dateien gespeichert und übertragen. Daraus resultiert, dass mittels der Alphanumerischen und der Orthographischen Zeichen einer Tastatur, keine komplexeren Textformatierungen vorgenommen werden konnten. Entscheidende Textpassagen, wie zum Beispiel Überschriften, Absätze oder Tabellen, mussten mittels einfacher Trennzeichen oder Zeilenumbrüche von einander abgegrenzt werden. Diese Vorgehensweise eignete sich nur bedingt um wissenschaftliche Informationen sinnvoll zu formatieren. Ein weiteres Problem bestand darin, dass die damaligen Rechnersysteme, zum Beispiel Atari, Commodore, PC, Apple oder Unix-Geräte, Daten unterschiedlich darstellten. Es war erforderlich den Inhalt einer Seite bzw. die reine Information von deren Darstellung zu trennen. Eine Sprache wurde benötigt, mit der die unterschiedlichen Seiteninhalte strukturiert werden konnten, aber keine genauen Angaben gemacht werden mussten, wie

diese Elemente darzustellen sind, darum sollte sich das jeweilige System kümmern.

Auf Grundlage dieser Erkenntnisse und der Erfahrung aus einem früheren Projekt, einem in Pascal geschriebenen Produktivitäts-Tool mit dem Namen „Enquire“, entwickelte der Informatiker Tim Berners-Lee 1989 am Kernforschungszentrum CERN in Genf - auf Grundlage von SGML - die Auszeichnungssprache HTML. Am 24. Dezember 1990 wurde die erste Webseite <http://info.cern.ch/> auf dem, ebenfalls von Berners-Lee entwickelten, ersten Webserver „CERN httpd“ in Betrieb genommen [web 1].

HTML beschreibt die Gliederung des Textes aber nicht dessen Darstellung, diese wird dem Browser auf dem jeweiligen System überlassen. Die notwendigen Elemente um Text zu strukturieren nennen sich „Tags“. Ein Textabschnitt, der eine Formatierung erhalten soll, wird von einem Starter-Tag und einem End-Tag eingeschlossen.

Einen weiteren und bedeutenden Vorteil brachte HTML mit sich, es konnte auf andere Inhalte oder Seiten „verlinkt“ werden. Sogenannte Hyperlinks machten es für Wissenschaftler möglich, Verweise auf andere wissenschaftliche Arbeiten einzubeziehen und somit die eigenen Arbeiten mit Informationen und Erkenntnissen anderer Wissenschaftler zu bekräftigen. Die Grundlage der Vernetzung war geschaffen und weitere HTML-Versionen folgten.

Diese neue Auszeichnungssprache war für Wissenschaftler eine enorme Bereicherung, aber für den privaten und kommerziellen Durchbruch fehlte noch etwas Entscheidendes. Der bis dahin übliche Austausch von Informationen, über die bestehenden Netzwerke, beschränkte sich auf die Darstellung von Text. Grüne Schrift auf schwarzem Hintergrund stellte, auch zur damaligen Zeit, ein visuell wenig anspruchsvolles Erlebnis dar. 1991 entwickelte Nicola Pellow, eine junge Mitarbeiterin und Mathematikerin am CERN, den ersten textbasierten-Browser<sup>10</sup>. Eine grafische Oberfläche, die als Mensch-

---

<sup>10</sup> <http://de.wikipedia.org/wiki/Webbrowser>

Maschine-Schnittstelle fungiert, die Grafiken und interaktive Elemente wiedergeben kann, wurde ein Jahr später mit den Browsern *Erwise*<sup>11</sup> und *ViolaWWW*<sup>12</sup> eingeführt. Im Jahr 1993 erschien zum einen der von Marc Andreessen und Eric Bina am NCSA entwickelte Browser *Mosaic* und zum anderen wurde die WWW-Technologie von seitens des CERN freigegeben. Die Freigabe ermöglichte eine Nutzung ohne Copyright- und Patentrechteverletzung und dies führte zu der raschen Verbreitung und Nutzung des WWW. Marc Andreessens Browser unterschied sich entscheidend von der Grundidee die Berners-Lee mit der Entwicklung von HTML verwirklichen wollte. Lee wollte die Manipulation der HTML-Seiten durch den Seiten-Besucher ermöglichen, wie es seit dem Web2.0-Zeitalter möglich ist. Mosaic und die folgenden marktbestimmenden Browser entsprachen dagegen reiner Lese-Software.

## 4.2 Entwicklungsschritte HTML 1.0 bis XHTML

Die Auszeichnungssprache HTML steht stellvertretend als Basis des World Wide Web. Seit ihrer Festlegung durch Tim Berners-Lee am 13. März 1989 wurde sie stetig weiterentwickelt. In erster Linie um mit der rasanten Entwicklung des Internet Schritt halten zu können und als Spezifikation für alle Browserhersteller zur Verfügung zu stehen. In diesem Kapitel werden die einzelnen Entwicklungsschritte näher beschrieben und am Ende ein Ausblick auf den sich anschickenden neuen Standard HTML5 gegeben.

---

11 <http://en.wikipedia.org/wiki/Erwise>

12 <http://en.wikipedia.org/wiki/ViolaWWW>

### **4.2.1 HTML 1.0**

HTML 1.0 ist mit dieser Versionsnummer nirgends explizit hinterlegt. Die Urversion von HTML, die am 03. November 1992 festgelegt wurde und sich nur am Text orientierte, entspricht fast der Version die Tim Berners-Lee am 13. März 1989 am CERN in Genf erfand und ist die erste offizielle Veröffentlichung von HTML. Die Möglichkeiten, die diese erste Version bot, waren sehr begrenzt, Text konnte nur durch Überschriften, Absätze und Verweise strukturiert werden. Fast exakt ein halbes Jahr später, am 30. April 1993, wurde eine neue HTML Version festgelegt, die um Schriftattribute wie fett und kursiv erweitert wurde und eine Bildintegration ermöglichte. Weitere sechs Monate später, im November 1993, wurden geplante Erweiterungen als HTML+ festgelegt, welche aber nie in die Urversion von HTML einfließen, sondern in späteren Versionen veröffentlicht wurden.

### **4.2.2 HTML 2.0**

Die Eigenschaften von HTML 1.0 wurde übernommen und durch grundlegende Elemente erweitert, um Dokumente besser strukturieren zu können. Ein Browser der die Grundelemente von HTML 2.0 nicht interpretiert, kann getrost als WWW-untauglich betrachtet werden. Faktisch wird HTML 2.0 als der aller-kleinste gemeinsame Nenner angesehen. HTML 2.0 war bereits Standard als der damalige Browser Primus „Netscape Navigator“ in der Produktversion 2.0 erschien, welcher bereits Frames und JavaScript - diese Elemente lagen weit über dem HTML-Standard -, beherrschte. Dadurch wurde HTML2.0 in der Internetgemeinde als Enttäuschung wahrgenommen. Bei der Erstellung von Seiten wurden lieber die neuen Möglichkeiten ausgereizt um Informationen

möglichst individuell darzustellen. HTML 2.0 wurde im November 1995 offizieller Sprachstandard und war es bis Januar 1997.

### 4.2.3 HTML 3.0

Im Jahr 1996 beginnt das Internet langsam das Interesse der Öffentlichkeit zu erreichen. Der Ruf nach mehr Gestaltungsfreiraum von HTML-Autoren wurde immer größer und dadurch waren die Entwickler von Netscape - der zu der Zeit bekanntesten Browsers - angehalten neue Tags und Attribute (Netscape Extension Tags) einzubauen.

Da allerdings andere Browser die für Netscape Navigator entwickelten Seiten nicht vernünftig darstellen konnten, entwickelte sich ein regelrechter Browserkrieg<sup>13</sup>. HTML-Entwickler mussten sich entscheiden, ob sie mit der aktuellen HTML-Version arbeiten oder für Netscape Navigator entwickeln wollten. Dieser Zustand war für ein sich entwickelndes Medium nicht zuträglich und die Suche nach einer Lösung begann. Eine Gruppe um Dave Raggett arbeitete an dem HTML 3.0-Entwurf, der viele Neuerungen und Verbesserungen in der Gestaltung versprach. Zu diesem Zeitpunkt hatten mehrere Browser Probleme mit der Umsetzung des aktuellen Standards. Nicht alle Elemente wurden implementiert, dafür kamen Browser-spezifische Elemente hinzu und diese ließen den HTML 3.0-Entwurf veraltet erscheinen, wodurch die Entwicklung dieser Spezifikation abgebrochen wurde.

---

<sup>13</sup> <http://server02.is.uni-sb.de/courses/ident/kontroverses/browserkrieg/>

### 4.2.4 HTML 3.2

Im Jahr 1994 wurde das W3C gegründet, um den Sinn und Zweck des Internets Rechnung zu tragen und eine weltweit gültige Spezifikation zu entwickeln. Das Internet sollte für jeden zugänglich gemacht werden und dazu war ein Standard, auf den sich die Browserhersteller stützen konnten, zwingend erforderlich. Die erste Veröffentlichung des W3C, mit dem Codenamen *Wilbur*, wurde schließlich HTML 3.2 benannt. Diese Version entsprach einer abgespeckten Version der geplanten HTML 3.0 -Version, zu der nur wenige neue Elemente hinzugefügt wurden. Größere Neuerungen wurden auf spätere Versionen verschoben, um nicht ausgereifte Schnellschüsse zu vermeiden. Über die Zeit entstandene Browsertags von Netscape oder Windows wurden aus den gleichen Gründen nicht in den Standard aufgenommen. Die veröffentlichte Version HTML 3.2 verbreitete sich schnell und wurde am 14. Januar 1997 zum offiziellen Standard erklärt. Nahezu alle heutigen Browser unterstützen diese Version vollständig.

### 4.2.5 HTML 4.0

Mit der am 18. Dezember 1997 veröffentlichten Version sind Stylesheets (CSS), Scriptsprachen und Frames eingeführt worden. Ebenfalls finden Tabellen, internationale Schriftzeichen und Multimediaobjekte mehr Unterstützung. Darüber hinaus wurden die drei Dokumenttypen *strict*, *transitional* und *frameset* hinzugefügt, die die Abgrenzung verschiedener Funktionalitäten und Anwendungsgebiete erleichtern sollen. Mit dieser Veröffentlichung bekräftigte das W3C, im mittlerweile zum Milliardenmarkt gewachsenen WWW, seine unabhängig Rolle, indem es sich mit den kommerziellen Entwicklungen auseinandersetzt und ihm große Bedeutung beimisst. Das Gremium

konnte nur überleben, weil es dadurch von den Softwareherstellern ernst genommen wurde, was für die Entwicklung des Internets ungeheuer wichtig war.

Eine leicht korrigierte HTML 4.0 Version erschien am 24. April 1998.

### **4.2.6 HTML 4.01**

HTML 4.01 stellt die letzte und aktuellste Überarbeitung der auf SGML basierenden HTML-Spezifikation dar. Sie wurde am 24. Dezember 1999 verabschiedet und trat eher lautlos an die Stelle von HTML 4.0. Die Überarbeitungen umfassen zahlreiche nachdrückliche Hinweise auf zukünftige Beschränkungen nicht erlaubter, aber aus Gründen der Abwärtskompatibilität geduldeter, Elemente. Des Weiteren wurde nachdrücklich darauf hingewiesen, dass der Inhalt eines Dokuments mittels Stylesheets (CSS) von dessen Präsentation getrennt werden soll.

### **4.2.7 XHTML**

HTML wurde entwickelt, um wissenschaftliche Informationen im Internet darzustellen und zu verbreiten. Von anfangs Text, Tabellen und Grafiken konnten im Laufe der Zeit immer mehr Elemente verwendet werden, z.B. Frames und Multimediaobjekte. Für das Internet war HTML ein großer Schritt in die Richtung der einheitlichen Wiedergabe auf verschiedenen Systemen, aber ein HTML-Dokument lies sich zum Beispiel nur in einem Browser korrekt darstellen. Drucken in geeigneter Form war auf Grund von fehlenden Randangaben nur auf Umwegen realisierbar. Der Tag-Vorrat ist begrenzt und Vermischungen von Struktur und Layout-Tags sind möglich.

In den ersten Jahren der Entwicklung wurde auf einen vollkommenen SGML-gerechten Aufbau nicht besonders viel Wert gelegt. Diese Tatsache führte dazu, dass HTML-Dokumente relativ Fehlertolerant sind und sich sehr schlecht von Programmen auf Fehler durchsuchen lassen.

Um die Probleme - sich über die Jahre angehäuft hatten - zu lösen, wurde zunächst XML entwickelt. Hauptziel der XML-Entwicklung war, einen Ausweg aus dem Dschungel der zahllosen Dateiformate, die sich nach einigen Jahren in der Szene entwickelt hatten, zu finden. XHTML ist eine Auszeichnungssprache die durch XML-Direktiven aufgebaut ist. Sie unterstützt nahezu alle Elemente von HTML 4.01 und ist durch beliebige Tags erweiterbar. XHTML 1.0 wurde am 26. Januar 2000 veröffentlicht. Der große Unterschied zwischen HTML und XHTML ist die SGML konforme Umsetzung. Dies bedeutet, dass ein XHTML-Dokument mit einem Programm auf Fehler durchsucht werden kann und bei ungültigen Angaben ein Fehler ausgegeben wird. Die Gültigkeit eines Dokuments kommt dem Gedanken, eine Sprache zu entwickeln, die es ermöglicht mit einheitlichen Methoden Dokumente auf unterschiedlichen Systemen zugänglich zu machen, viel näher, als es HTML bis dato je gelungen war. Die am 31. Mai 2001 veröffentlichte XHTML 1.1-Version ist nicht wie XHTML 1.0 in drei Dokumententypen unterteilt, sondern nur *strict* wird unterstützt. Eine weiterführende Version mit dem Namen XHTML 2.0 wurde in der Entwicklungsphase zu Gunsten von HTML5 eingestellt.

### 4.3 HTML5 – Der aktuelle Stand der Entwicklungen

Die in der Entwicklung befindliche HTML5-Spezifikation stellt eine Weiterentwicklung der bestehenden Standards HTML 4.01, XHTML 1.0 und DOM2 dar. Sie steht als Oberbegriff für die internen Bezeichnungen HTML5, XHTML5, DOM5 und damit für neue Techniken mit denen Webseiten und Medien-Inhalte effizienter zur Verfügung gestellt werden können.

Mitte des Jahres 2004 wurde von der WHATWG der erste Vorschlag für HTML5 veröffentlicht. Der erste Arbeitsentwurf wurde am 22. Januar 2008 und der vorerst letzte am 13. Januar 2011 herausgebracht. Ein Arbeitsentwurf fasst den zum Veröffentlichungszeitpunkt aktuellen Entwicklungsstand zusammen. Die Aufgabe die sich das W3C mit der Entwicklung von HTML5 und den damit verbundenen Zielsetzungen gestellt hat, ist nicht leicht umzusetzen.

Die neue Sprachspezifikation soll Kompatibilität zu bestehendem Inhalt bieten, neue Funktionen zur Verfügung stellen, die echte Probleme lösen und den Sicherheitsaspekt mehr denn je beachten, den Entwicklungsprozess vereinfachen und den nächsten Schritt zum semantischen Web darstellen. Diese Schwerpunkte miteinander zu verknüpfen ist schwer, denn HTML5 soll zusätzlich auf allen Endgeräten mit jeder Weltsprache verwendbar sein, definiertes Verhalten in allen Fehlersituationen und geringe Komplexität bieten. Darüber hinaus ist HTML5 die erste HTML-Version in der für Browserhersteller definiert wird, wie der HTML-Parser zu arbeiten hat und wie Fehler und andere browserspezifische Dinge behandelt werden sollen. Die Zielsetzung ist ein umfassendes Konzept, das für die Zukunft gerüstet ist.

Für jedes Element wird beschrieben wo es im Dokument stehen darf, über welche Attribute es verfügt, was das Element für eine Aufgabe erfüllt und wie die DOM-Schnittstelle definiert ist. Zusätzlich ist ein Teil enthalten, der die browserspezifische

Verarbeitung und detaillierte technische Ausführungen die für Browserhersteller relevant sind, beschreibt. Kurz gesagt, wird es mit dieser Spezifikation für einen versierten Programmierer möglich sein, einen Browser zu programmieren, der HTML5-Dokumente darstellen kann.

Trotz dieser großen Bestrebungen wird es noch Jahre dauern, bis alle Browserhersteller diesen neuen Standard vollständig in ihr Produkt integriert haben. Viele der aktuell erhältlichen Browser unterstützen bereits HTML5-Elemente, aber das sind nur Ausschnitte aus HTML5 und im Endeffekt Vorabversionen der jeweiligen Funktionalität.

Der Marketingkampf unter den Browserherstellern wird ziemlich hart geführt. Von diesem Aspekt aus betrachtet, ist es nicht verwunderlich, wenn schnell mal eben das eine oder andere „neue“ Element in der folgenden Version Platz findet. Oftmals mit dem Wissen, dass der wirkliche Nutzen dieses Elements momentan noch nicht ersichtlich ist. Dieses Vorgehen führt dazu, dass jeder Hersteller ein Element auf eine etwas andere Art integriert, weil der Standard bekanntlich noch nicht existiert.

Abgesehen von den beschriebenen Aspekten der Integration in den Browser und der allgemein umfassenden Spezifikation werden in HTML5 nützliche Funktionalitäten, wie das Canvas-Element, Multimedia-Embedding, Offline-Storage/Speicherung und semantisches Markup, eingeführt.

### 4.3.1 Das Canvas-Element

Das Canvas-Element, zu deutsch „Leinwand“, stellt eine Fläche zur Verfügung auf die mittels JavaScript dynamische Bitmap-Grafiken gezeichnet werden können. Es handelt sich im Prinzip um ein programmierbares `<img>`-Element. Als besonderer Vorteil, z.B. gegenüber Flash und Silverlight, ist eine einfache Integration in eine HTML-Seite und Kommunikation mit selbiger zu nennen. Ob diese Argumente gegenüber Flash mit seinem *ExternalInterface* in Kombination mit JavaScript als so großer Vorteil zu betrachten sind ist fraglich. Dafür können beliebige HTML-Elemente über dem Canvas platziert werden, was bei einer SWF-Datei nicht ohne weiteres möglich ist.

### 4.3.2 Das embed-Element

Ein schon lange in der nicht standardisierten Praxis benutztes Tag ist *embed*<sup>14</sup>. Es wurde in die Spezifikation aufgenommen und ist genauso zu benutzen wie bisher. Über das *src* Attribut wird der Pfad zu einer Medien-Datei, mit *type* die Art des Inhalts mit dem entsprechenden MIME-type angegeben und über *width* und *height* die Größe definiert. Zusätzlich können relevante Werte, wie z.B. *allowscriptaccess*=*“sameDomain“* oder *allowfullscreen*=*“true“* angegeben werden. Ein Nachteil ist, dass das *embed*-Tag - ein sogenanntes „Leer“-Tag - keinen Ersatzinhalt aufnehmen kann, was z.B. benötigt wird, wenn das Flash-PlugIn auf dem Client-System nicht installiert ist. Abhilfe kann an dieser Stelle das *object*-Tag schaffen. Mit ihm lassen sich alle Arten von externen Inhalten einbinden. Die Nutzung von *object* ist identisch mit der von *embed*, aber *object* ist kein „Leer“-Tag. Dadurch lässt sich zwischen dem öffnenden und schließenden Tag Ersatzinhalt, wie zum Beispiel eine Information zum fehlenden Flash-PlugIn, angeben.

<sup>14</sup> <http://de.selfhtml.org/html/multimedia/netscape.htm>

Ein wirklicher Mehrwert des *embed* gegenüber dem *object*-Tag ist im Zusammenhang mit der Einbindung einer SWF-Datei nicht zu erkennen. Möglich wäre, dass sich bis zur Veröffentlichung des finalen Standards daran noch etwas ändert. Der aktuelle Stand ist diesbezüglich nicht ganz klar abgegrenzt.

### 4.3.3 Die Elemente audio und video

In Bezug auf Multimediaunterstützung machen die neuen Tags *audio* und *video* den großen Vorteil von HTML5 gegenüber der aktuellen HTML 4.01 Spezifikation aus. Die Vormachtstellung, die Adobe mit Flash in diesem Segment inne hat - es gibt nur wenige andere Möglichkeiten Audio- oder Video-Inhalte in eine Web-Seite zu integrieren - soll damit durchbrochen werden.

Mit dem *audio*-Tag lassen sich Sounds und Audiostreams einbinden. Mit maximal drei Zeilen Code ist ein minimaler aber funktionstüchtiger Audioplayer fertiggestellt. Das *audio*-Tag verfügt, im Gegensatz zu *video*, über eine Constructor-Funktion. Dadurch besteht die Möglichkeit bei einem Event, zum Beispiel einem Klick auf einen Button, direkt einen Sound abzuspielen.

Dies würde wie folgt aussehen: `new Audio( 'audio.mp3' ).play();`

Mit dem *video*-Tag lassen sich Video-Inhalte progressiv wiedergeben. Dies bedeutet, die Videodatei wird vom Server auf die Festplatte des Client-Systems heruntergeladen und steht, nachdem eine definierte Pufferzeit geladen wurde, zum Abspielen bereit.

Das „streamen“ von Video-Inhalten ist mit HTML5 momentan noch nicht möglich. Als eine Ursachen dafür ist die nicht geklärte Codec-Frage zu nennen. Sie trägt maßgeblich Schuld daran, dass es in diesem Bereich keine großen Fortschritte gibt. Die

Entwicklung eines einheitlichen Streaming-Protokolls - eine Grundvoraussetzung - ist unter den aktuellen Umständen nahezu unmöglich.

Ein weiteres Problem stellt die Wahrung der Urheberrechte von Video-Inhalten dar. HTML5 bietet dafür bis dato keine Lösung. Wer sich eine Kopie eines Videos erzeugen will, kann den Quelltext der Web-Seite öffnen, den Link zur Video-Datei kopieren und diese anschließend downloaden.

Bei beiden Elementen kann zwischen dem öffnenden und dem schließenden Tag Ersatzinhalt platziert werden, der angezeigt wird, wenn das Element vom Browser noch nicht unterstützt wird. Des Weiteren lässt sich das Erscheinungsbild der beiden Elemente, zum Einen die Größe über die Attribute *width* und *height* und zum Anderen mit CSS an das Layout der Webseite anpassen und somit eine nahtlose Integration sicherstellen.

Folgende Attribute werden von beiden Elementen verarbeitet. Das *src*-Attribut nimmt eine Pfadangabe zu der abzuspielenden Datei entgegen. Das *controls*-Attribut legt fest, ob Steuerungsbuttons angezeigt werden oder nicht. Die automatische Wiedergabe kann mit dem *autoplay*-Attribut festgelegt werden und ob die Medien-Datei als Endlosschleife abgespielt werden soll, wird mit dem *loop*-Attribut definiert. Als Wert für die drei Attribute *controls*, *autoplay* und *loop* ist „true“ oder „false“ möglich.

Bei der Verwendung des *video*-Tag kann zusätzlich zu allen anderen Attributen das *poster*-Attribut angegeben werden. Es nimmt einen Pfad zu einer Grafik entgegen, die angezeigt wird solange das Video noch nicht gestartet wurde.

Die Nutzung der beiden Elemente ist ziemlich einfach. Als erstes wird die Quelldatei, dann die Breite und Höhe, die gewünschten Wiedergabeoptionen und zwischen öffnendem und schließendem Tag möglicher Ersatzinhalt angegeben. Diese handvoll Angaben reichen aus, um mit beiden Elementen einen Player zur Verfügung zu stellen.

Nun kann leider nicht jeder Browser mit jedem Codec umgehen. Um dem Browser

mehrere Varianten einer Quell-Datei zur Verfügung zu stellen, gibt es in beiden Tags das *source*-Element, welches zwischen öffnendem und schließendem Tag als Liste eingefügt wird. Das Element verfügt über das *src*-Attribut zum angeben des Dateipfades und kann zusätzlich die Attribute *media* und *type* verarbeiten. Mit dem *media*-Attribut kann der Typ des über *src* angegeben Inhalts, oder das zu favorisierende Endgerät (*media*="handheld" oder *media*="all") angegeben werden. Über das *type*-Attribut kann dem Browser mitgeteilt werden, um welche Art von Datei es sich bei der Quell-Datei handelt. Mögliche Werte wäre zum Beispiel „audio/mp3“ und „video/ogg“. Des Weiteren kann diese Attribut den optionalen Parameter *codecs* aufnehmen, mit dem als erstes der Video-Codec und als zweites der Audio-Codec angegeben werden kann. Eine Implementation würde wie folgt aussehen:

```
<source src="video.ogg" type="video/ogg; codecs='theora, vorbis'">
```

Mittels dieser Attribute kann der Browser feststellen, ob er diesen Dateityp überhaupt verarbeiten kann. Dadurch wird verhindert, dass für jede Mediendatei der Ladevorgang begonnen werden muss, um Metadaten zu erhalten, aus denen ermittelt werden kann, ob eine Verarbeitung möglich ist. Unnötige Wartezeiten für den Nutzer können dadurch minimiert werden. Die angelegte Liste der *source*-Elemente wird vom Browser nacheinander durchgegangen und die erste Quelldatei, die er verarbeiten kann, wird wiedergegeben. Theoretisch kann über das erläuterte *source*-Element ein Video in jedem Codec angelegt werden und die Wiedergabe in allen Browsern wäre sichergestellt. Aber es macht schlicht keinen Sinn ein Video in alle möglichen Formate zu konvertieren. Besser wäre die Lösung, alle HTML5 fähigen Browser würden ein Codec unterstützen, womit die Wiedergabe in allen Browsern gewährleistet wäre. Aber an dieser Stelle gibt es das bekannte Problem, dass sich die Browserhersteller noch auf keinen gemeinsamen Codec einigen konnten.

### 4.3.4 Die Media-API

Für die Steuerung der neuen Media-Elemente steht die gemeinsame Media-API zur Verfügung. Der einzige Unterschied zwischen den beiden Elementen ist, dass das *video*-Element über die Attribute *videoWidth* und *videoHeight* die Größe des aktuellen Elements zurück gibt. Ansonsten sind die zur Verfügung gestellten Funktionen identisch. Unter anderem stehen folgende Funktionen zur Verfügung, mit denen auf einfache Weise ein Player mit eigenen Steuerungs-Schaltflächen erstellt werden kann.

- play()* - Startet die Wiedergabe
- pause()* - Stoppt die Wiedergabe
- onplay()* - Aufruf wenn Wiedergabe gestartet wurde
- onpause()* - Aufruf wenn Wiedergabe gestoppt wurde
- duration* - Gesamtzeit der Mediendatei
- currentTime* - aktuelle Zeit
- onerror()* - Aufruf wenn ein Fehler auftritt

### 4.3.5 Offline Web Application API

Ein sehr interessanter Bestandteil der neuen Spezifikation ist die „Offline Web Application API“. Mit ihr hat der Webentwickler die Möglichkeit, eine Web-Applikation „offline“-tauglich zu erstellen. Mittels einer Cache-Manifest-Datei wird dem Browser mitgeteilt, welche Ressourcen beim Aufruf der Seite vorzuladen sind (gecached werden sollen) damit die gewünschte Funktionalität bei Verlust der Internetverbindung weiterhin zur Verfügung steht. Unter den Ressourcen können sich ganze HTML-Seiten, Bilder und Skripte befinden. Dadurch sollen unter anderem offline durchgeführte Nutzeraktionen, zum Beispiel ein neuer Datenbank-Eintrag oder das Absenden von Formulardaten zwischengespeichert werden und die Übertragung erfolgt beim nächsten Onlinegang automatisch.

### 4.3.6 Semantisches Markup

Mit HTML5 wird eine Vielzahl neuer Elemente eingeführt, mit denen sich Dokumente semantisch auszeichnen lassen, worunter das so genannte semantische Markup zu verstehen ist. Die semantische Auszeichnung einer Web-Seite ist dahingehend wichtig, dass eine Suchmaschine dem verschiedenen Inhalt eine Gewichtung zuordnen kann und dafür sind aussagekräftigere Elemente notwendig. Ein schlechtes Beispiel ist das `div`-Tag, in dem sich nahezu alles befinden kann und die Suchmaschine hat keine Möglichkeit einen Unterschied festzustellen. Elemente wie *section*, *header*, *footer*, *nav*, *aside*, *article*, die Überschriften-Elemente *h1* bis *h6* mit dem neuen Outline-Algorithmus und *hgroup* legen die grobe Struktur eines Dokuments fest und geben Informationen preis, welcher Bereich was für einen Nutzen hat. Darüber hinaus gibt es

neue, alte (mit Modifikationen) und reaktivierte Elemente zur semantischen Textauszeichnung [Kroener 2010].

### 4.3.7 Neue Formular Elemente

Alle Eingabeelemente bieten neue Input-Typen, neue Funktionen und sind mit der neuen Validierungs-API verbunden. Letztere bietet die Funktionalität, die vor HTML5 mit JavaScript-Bibliotheken umständlich integriert werden musste. Dies erübrigt eine Menge Programmieraufwand und führt zu einem übersichtlichen und schlanken Code. Als Input-Typen sind *email*, *url*, *number* und *date* zu nennen. Die Input-Typen *email* und *url* lassen zum Beispiel nur korrekte E-Mail-Adressen bzw. URLs als valide gelten. In Bezug auf eine mögliche Validierung dieser beiden Typen ist das *multiple*-Attribut zu nennen, es ist aber auch bei anderen Elementen verfügbar. Wird es angegeben, können mehrere E-Mail-Adressen bzw. URLs in ein Eingabefeld eingegeben werden und jede wird für sich auf Korrektheit geprüft. Dadurch sind mit wenigen Zeilen Code relativ umfangreiche Eingabeformulare realisierbar, abgesehen von dem jeweiligen Code auf der Serverseite.

Diese genannten Funktionalitäten werden mit ziemlicher Sicherheit in einem finalen HTML5-Standard Platz finden und sind in den aktuellen Browsern teilweise integriert und testbar<sup>15</sup>. Leider werden die sich durch die neuen Elemente resultierenden Aufwandseinsparungen erst rentieren, wenn HTML5 veröffentlicht und in allen Browsern integriert ist. Bis dahin, muss bei vielen Elementen - der Browser-Integrationen geschuldet - ein Umweg eingeschlagen werden.

---

<sup>15</sup> <http://html5test.com/>

### 4.4 Einsatzbereiche

Der Einsatzbereich von HTML5 ist das Internet und in erster Linie der mobile Anwendungsbereich. Mobile Endgeräte sind wie geschaffen um neue HTML5-Funktionen zum Einsatz zu bringen. Dabei geht es nach der Devise, was im kleinen funktioniert erfüllt auch im großen (Desktop) seinen Zweck.

Das Canvas-Element in Kombination mit JavaScript findet Verwendung für eindrucksvolle Effekte und kleine Spiele<sup>16</sup>. Die Geolocation-Klasse<sup>17</sup> (JavaScript-API) ermöglicht die Berechnung der genauen Position mittels GPS, WiFi oder IP-Auswertung.

Mit Web-Storage<sup>18</sup> (sessionStorage und localStorage) ist es möglich größere Datenmengen auf dem Client-System zu speichern und dadurch während eines Verbindungsverlustes die Funktionalität der Anwendung aufrecht zu erhalten.

Voraussetzung ist immer ein moderner Browser, den moderne mobile Endgeräte (Smartphone) in Form einer speziellen mobil-Version (ausgenommen Apples Safari) aufgrund ihrer Aktualität serienmäßig an Board haben [web 2].

Im Desktop-Bereich sollte, solange die Entwicklungsarbeiten an der neuen Spezifikation andauern, sehr sorgsam abgewägt werden, ob der Einsatz zweckmäßig und sinnvoll ist.

---

16 <http://savedelete.com/best-html5-canvas-games.html>

17 <http://9elements.com/html5demos/geolocation/>

18 <http://www.braekling.de/web-design/3596-html5-web-storage-local-und-session-storage.html>

### 4.5 Vor- und Nachteile

Die Vorteile von HTML5 gegenüber dem aktuellen HTML Standard sind vielfältig. Die in Kapitel 4.3.3 vorgestellten neuen Elemente bieten Funktionalität, die bis dato mit externen Bibliotheken integriert werden mussten. Darüber hinaus stellt die offline Unterstützung für mobile Endgeräte einen enormen Vorteil dar. Das semantische Markup ermöglicht eine bessere Suchmaschinenindexierung ohne viel Zutun des Programmierers. Die neuen Multimedia-Elemente bieten mit wenig Zeilen Code relativ viel Funktionalität. Im Allgemeinen sind die neuen Elemente mit kurzen Anweisungen integriert und halten so den Quellcode schlank und übersichtlich. In diesem Zusammenhang kann auch der neue Doctype `<!DOCTYPE html>` genannt werden, der sich leicht merken lässt und die verschiedenen Modi der aktuellen Versionen überflüssig macht.

Eine Besonderheit stellt das Verhalten von HTML5-Elementen dar. Mittels Graceful Degradation (ansprechende Herabsetzung) wird der Browser in die Lage versetzt, neue Elemente zu ignorieren, wenn er diese nicht kennt. Die Erweiterung dieses Verhaltens ist Progressive Enhancement (progressive Verbesserung), wodurch anstelle eines neuen Elements ein vergleichbares altes Element mit herkömmlicher Implementation zur Verfügung gestellt wird.

Als klarer Nachteil muss die Struktur und Arbeitsweise der zwei Gremien W3C und WHATWG angesehen werden. Auf Veränderungen am Markt kann nicht schnell genug reagiert werden. Die bestehenden Strukturen verlangen, dass sich die Partner untereinander einigen, welche Techniken und Grundlagen in die Spezifikation aufgenommen werden. An der Festlegung eines HTML5 Video-Codec ist diese Problematik sehr deutlich zu erkennen. Der Entscheidungsprozess ist in der Praxis langatmig und unflexibel, wodurch nur relativ schlecht auf Marktveränderungen reagiert

werden kann. Abhilfe könnte die im Abschnitt 4.6 genannten Überlegungen zu einem versionsunabhängigen HTML Standard bieten. Dadurch könnten einzelne Neuerungen viel einfacher und schneller veröffentlicht werden.

### 4.6 Ausblick

Seit Mai 2011 läuft der sogenannte LastCall<sup>19</sup>, der den letzten Abschnitt des Entwicklungsprozesses darstellt. Diese Phase wird dafür genutzt, Rückmeldungen zum aktuellen Entwicklungsstand einzuholen und diese noch einmal eingehender zu begutachten. Die Fertigstellung der Spezifikation ist für das Jahr 2014 geplant.

HTML5 wird ohne Zweifel eine Veränderung bewirken. Auf mobilen Endgeräten wird es schon vermehrt eingesetzt und ist aus diesem Segment nicht mehr wegzudenken. Wie schnell die Verbreitung in normalen Webseiten voranschreiten wird, ist schwierig abzuschätzen. Teile der Spezifikation werden schon verwendet, vieles ist noch im unklaren. Es ist durchaus vorstellbar, dass der durchschnittliche Nutzer den Wechsel vom aktuellen zum neuen Standard gar nicht bemerkt. Einzig die Programmierer und Interessierte werden einen Unterschied bemerken.

Ob HTML5 Flash verdrängen wird ist zu bezweifeln. Die neuen Multimedia-Elemente stellen von den Möglichkeiten und der Usability keine direkte Konkurrenz für Flash dar, aber können es in einfachen Anwendungen durchaus ersetzen. Sollten die diversen neuen Elemente irgendwann in allen Browsern integriert sein und deren Implementation dem Standard entsprechen, wird sich dadurch für Programmierer eine enorme Aufwandsersparnis einstellen. Die diversen Browser- und Versionsunterscheidungen würde entfallen und zu einem sauberen und einfacher zu erweiternden Quellcode

---

<sup>19</sup> <http://www.w3.org/2011/02/htmlwg-pr.html>

#### *4 HTML5*

---

führen. Die Diskussionen gehen unterdessen schon über HTML5 hinaus. Die WHATWG zielt auf ein Entwicklungsmodell ohne Versionsnummer, dessen Teilbereiche sich in unterschiedlichen Entwicklungsstadien befinden können [web 3].

Die Vorteile liegen klar auf der Hand. Der ewig Lange Prozess zu einer neuen Version wäre überflüssig. Einzelne Teilaspekte könnten aktualisiert und Neuheiten veröffentlicht werden. In diesem Szenario müssten die Browser-Hersteller den Standard bedienen und nicht wie momentan, das die Browser Funktionalität unterstützen, die es offiziell noch gar nicht gibt.

## **5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung**

In diesem Kapitel findet der Vergleich der in den vorangegangenen Kapiteln vorgestellten Technologien Flash und HTML5 statt. In diesem Zusammenhang wird eine Anwendung mit zwei Frameworks entwickelt und Schritt für Schritt anhand des MVC-Entwurfsmusters gegenüber gestellt. Dazu wird zunächst die Anwendung und das zu Grunde liegende MVC-Grundmuster vorgestellt. Anschließend werden nacheinander die MVC-spezifischen Bestandteile der Anwendung beschrieben, auf eventuelle Gemeinsamkeiten und Unterschiede eingegangen und eine Schlussfolgerung abgegeben. Im Anschluss erfolgt eine Auswertung der vorangegangenen Gegenüberstellung betreffend der angestrebten Zielstellung und ein abschließendes Fazit.

### **5.1 Übersicht**

Dieses Kapitel fasst alle für den Vergleich notwendigen Basisinformationen zusammen, stellt die zu programmierende Anwendung vor und zeigt die Gründe für den Vergleich und die daraus resultierenden Zielstellung auf.

Die Web-Technologien Flash und HTML5 sind nicht direkt vergleichbar. Flash ist eine Entwicklungsumgebung, die das Erstellen interaktiver Anwendungen ermöglicht.

HTML5 hingegen steht als Oberbegriff für verschiedene Technologien und soll die nächste Version der Seitenbeschreibungssprache HTML werden. Die technologischen Grundsätze unterscheiden sich zu stark, als dass ein direkter Vergleich durchgeführt

werden kann. Eine SWF-Datei muss in ein HTML-Grundgerüst eingebunden werden und wird vom Browser-PlugIn, dem Flash Player, abgespielt. Daraus resultiert, dass Flash von HTML abhängig ist, denn ohne HTML kann eine SWF-Datei nicht in eine Webseite angezeigt werden. Aus diesem Grund ist ein Vergleich nur in Kombination mit einem Framework und einer Anwendung, die mit beiden Technologien umsetzbar ist, sinnvoll. Die gewählten Frameworks setzen beide das MVC-Entwurfsmuster vollständig um und bieten dadurch eine ideale Basis für eine Gegenüberstellung.

### **5.1.1 Die Anwendung**

Die zu entwickelnde Anwendung ist ein Verwaltungswerkzeug für das gemeinsame Nutzen von Gebrauchsgegenständen, woraus der Anwendungsname „ShareIt“ abgeleitet wurde. Registrierte Personen können Tauschgegenstände anlegen und zum Tauschen freizugeben. Verfügbare Tauschobjekte sind in einer Liste auswählbar und über Schaltflächen kann Kontakt zum Eigentümer aufgenommen werden. Weitere Optionen wie z.B. Weiterempfehlen, Kontakt zum aktuellen Besitzer oder Reservierung eines Nutzungszeitraums sind denkbar. Darüber hinaus kann der Nutzer die Liste der Tauschobjekte filtern, um nur seine eigenen, Beteiligungen an anderen oder aktuell in seinem Besitz befindlichen Tauschobjekte angezeigt zu bekommen. Die registrierten Nutzer können ebenfalls in einer Liste angezeigt werden.

In der prototypischen Anwendung „ShareIt“ wird nur eine Basisfunktionalität des soeben vorgestellten Anwendungsumfangs integriert. Diese Restriktionen ermöglichen die Entwicklung von überschaubaren und vergleichbaren Programmcode.

Basisfunktionalitäten:

- Registrieren
- Anmelden
- Übersicht/Erklärungstext
- Übersicht über alle Tauschobjekte
- Filterung der Tauschobjekte nach Beteiligung, eigenen Tauschobjekten und aktuell im Besitz befindliche Tauschobjekten
- Registrieren eines neuen Tauschobjektes
- Übersicht aller registrierten Personen
- Aktualisierung der Nutzerdaten
- Abmelden

Je nach Entwicklungsfortschritt werden die Basisfunktionalitäten in beiden Prototypen integriert. Sollte die Zeit zur vollständigen Implementation nicht ausreichen, werden die Funktionen und Folgeaktionen im Quellcode angedeutet.

### **5.1.2 Ziel des Vergleichs**

Der angestrebte Vergleich soll evaluieren, welche der zu vergleichenden Techniken die bessere Grundlage bietet um eine Anwendung, wie im Abschnitt 5.1.1 beschrieben, zu entwickeln.

### **5.1.3 Nicht betrachtete Aspekte**

In diesem Vergleichsszenario wird die Serverseite einer Anwendung bewusst nicht behandelt, da die verwendeten Technologien im Browser und dadurch auf der Clientseite ausgeführt werden und nur im geringen Maß die Unterstützung eines Servers benötigen. Dies wird besonders an dem in Abschnitt 5.2.2 vorgestellten Framework deutlich und lässt sich auf Adobes Flash-Technologie adaptieren.

## **5.2 Verwendete Frameworks**

Für diesen Vergleich wird seitens Flash die aktuellste Version der Programmiersprache ActionScript 3.0 und ein Architektur-Framework, das das MVC-Architektur-Muster vollständig umsetzt, eingesetzt. ActionScript 3.0 ist eine ausgereifte und umfangreiche Programmiersprache und eignet sich für nahezu jegliche Art von Anwendung. In Kombination mit einem MVC-Architektur-Muster lassen sich komplexe Anwendungen (RIA) strukturiert und effizient entwickeln.

Seitens HTML5 wird ein neues Framework verwendet, das als erstes seiner Art betrachtet werden kann. Es vereint mit der Programmiersprache JavaScript das MVC-Architektur-Muster und diverse Technikspezifikationen des neuen HTML5 Standards. Dadurch wird eine neue Art der Anwendungsentwicklung im Internet ermöglicht.

### **5.2.1 Flash Framework PureMVC**

PureMVC wurde von Clifford Hall entwickelt, um ein leichtgewichtiges und dennoch leistungsstarkes Gerüst zum erstellen reichhaltiger Web-Applikationen (RIA) mit ActionScript zur Verfügung zu stellen. Hall richtete sich bei der Entwicklung maßgeblich an die Vorgaben aus dem Buch der „Gang of Four“<sup>20</sup> zur Entwicklung mittels Entwurfsmuster.

Die Logik einer Anwendung wird durch die Verwendung des PureMVC-Framework von Anfang an in drei eigenständige Bereiche, die Hauptakteure Model, View und Controller, unterteilt. In PureMVC sind dies sogenannte Singletons (eine Klasse von der zur Laufzeit nur eine Instanz erzeugt wird). Ein viertes Singleton, die Facade, stellt die Hauptinstanz von PureMVC dar und initialisiert die anderen Singletons zur Laufzeit. Die PureMVC Facade wird durch eine „konkrete“ Facade abgeleitet. Dadurch erhält die sie Zugang zu allen Hauptakteuren, ohne mit diesen direkt in Kontakt zu stehen. Die konkrete Facade wird als Referenz in der Notifier-Klasse angelegt und ist aufgrund der Vererbung in jedem Hauptakteur zugänglich. Dadurch wird der Zugriff auf Hauptfunktionalitäten der Facade und das Nachrichtensystem von PureMVC, die Kommunikation mittels Notifikations, ermöglicht. Jeder Mediator kann mit der Funktion 'listNotificationInterests()' (Rückgabewert ist ein Array) Interesse an bestimmten Notifications bekunden. Wird eine Notification gesendet, erhalten alle Mediatoren diese Notification, die an ihr interessiert sind. Dahinter verbirgt sich das Observer (Beobachter) -Muster. Der dadurch zur Verfügung stehende Veröffentlichen-/Abonnieren-Mechanismus ermöglicht es, dass beliebig viele Beobachter die gleiche Notification empfangen können und gewährleistet eine lose Kopplung von Sender und Empfänger.

---

<sup>20</sup> „Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software“ - ISBN 3-8273-2199-9

## *5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung*

---

Alle drei Hauptakteure können Notifications versenden, aber nicht alle können sie empfangen. Der Akteur Proxy wäre zu stark mit View bzw. Mediator und Controller verbunden, wenn er auf Notifications reagieren könnte. Die Aktualisierung der Daten eines Proxies, z.B. nach einem Nutzer-Ereignis, sollte der zuständige Mediator bzw. Controller übernehmen.

Die Controller-Schicht, die Fachlogik der Anwendung, wird in PureMVC vollständig von Commands, die auf Notifications gemappt werden können, implementiert. Es stehen zwei Varianten von Command zur Verfügung. Ein SimpleCommand enthält Anwendungslogik, kann Proxies und Mediators abfragen, manipulieren und ggf. entfernen und Notifications versenden. Dadurch kann ein SimpleCommand mit allen wichtigen Akteuren interagieren und Einfluss auf die komplette Anwendung nehmen. Ein MacroCommand kann mit `'addSubCommand()'` beliebig viele SubCommands (SimpleCommand oder MacroCommand) hinzufügen, die nacheinander abgearbeitet werden. Jedem SubCommand wird beim Aufruf eine Referenz der gesendeten Notification übergeben.

Um PureMVC in einem Projekt einzusetzen, kann das Framework als swc-Datei in das Projekt importiert, oder als Klassenverzeichnis, um Zugriff auf die verschiedenen Basisklassen zu erhalten, in die Verzeichnisstruktur integriert werden. Vorher muss eine der zwei Versionen (Standard<sup>21</sup> oder Mehrkern<sup>22</sup>) gewählt werden. Die Standard-Version entspricht einem Kern, einer Facade die Model, View und Controller kapselt. Die Mehrkern-Version ermöglicht es, mehrere Kerne in einer Anwendung zu benutzen. Für die prototypische Entwicklung ist die Standard-Variante ausreichend.

---

21 [http://trac.puremvc.org/PureMVC\\_AS3/wiki/Downloads](http://trac.puremvc.org/PureMVC_AS3/wiki/Downloads)

22 [http://trac.puremvc.org/PureMVC\\_AS3\\_MultiCore/wiki/Downloads](http://trac.puremvc.org/PureMVC_AS3_MultiCore/wiki/Downloads)

### **5.2.1.1 Funktionsumfang**

Das PureMVC-Framework ist ein reines Architektur-Framework und stellt außer die Kapselung der Singletons Facade, Model, View und Controller, Commands und dem Kommunikationsmittel Notifications keine weitere Funktionalität zur Verfügung.

In Kombination mit ActionScript 3.0 stehen dem Programmierer aber alle Mittel bereit, eine reichhaltige Web-Anwendung zu entwickeln. Die ActionScript interne Kommunikation erfolgt mittels Events. Daten können in beliebiger Form gespeichert und geladen werden und die Strukturierung der Anwendung ist mittels Vererbung auf unterschiedlichste Weise möglich.

Die drei MVC-Akteure werden jeweils noch einmal unterteilt, um eine strikte Trennung von Struktur und Logik zu gewährleisten.

Die Model-Schicht wird unterteilt in Model und Proxy.

Ein Model legt die Datenstruktur eines abzubildenden Objektes fest. Ein Proxy kommuniziert mit Remote-Services und kann eine Instanz eines Models mit Daten füllen. In PureMVC stellen Models in Kombination mit Proxies ein Schema dar.

Die View-Schicht wird unterteilt in View und Mediator.

Die View enthält eine Referenz zu einem Mediator und ausschließlich die Definition des darzustellenden Erscheinungsbildes und Funktionalität, die dieses aufrecht erhält.

Der Mediator beherbergt die Logik der View und sendet bzw. empfängt Notifications, um den Zustand der View auf dem aktuellen Stand zu halten.

Die Controller werden in PureMVC vollständig von Commands implementiert. Mit der

Funktion `'registerCommand()'` kann ein Notification-Name auf ein Command gemappt werden. Ein Command kann Proxies und Mediatoren abfragen, diese manipulieren und ggf. entfernen. Darüber hinaus kann ein Command weitere Notifications versenden. Wurde ein Command mit der Funktion `'registerCommand()'` mit einem Notification-Namen verknüpft und egal in welchem Teil der Anwendung gesendet, wird das entsprechende Command ausgeführt. Demzufolge wartet ein Command nicht auf eine Notification, es wird erstellt und ausgeführt, wenn es benötigt wird. Nach der Abarbeitung ist es nicht mehr existent.

Das Framework Kommunikationsmittel heißt Notification und kann von allen drei Akteuren versendet werden. Über die interne Funktionsweise von Notifications braucht sich der Programmierer keine Gedanken zu machen. Um eine Notification zu versenden, wird die vererbte Funktion `'sendNotification()'`, die bis zu drei Parameter entgegen nehmen kann, verwendet. Der erste Parameter ist der Notification-Name (String), danach folgen die optionalen Parameter `'body'` (Object) und `'type'` (String). `'body'` bietet die Möglichkeit beliebige Daten zu versenden. Mit `'type'` kann eine zusätzliche Zustandsunterscheidung beim Empfänger erfolgen.

### **5.2.1.1 Vorteile**

Aus Sicht der Anwendungsstrukturierung bietet PureMVC mit ActionScript 3.0 alles was für ein umfangreiches Projekt nötig ist. Das Framework stellt den de facto Standard für die Entwicklung mit Flash oder Flex dar und ist absolut neutral gegenüber der verwendeten Technologie. Die Trennung der MVC-Hauptakteure ermöglicht eine modulare und enorm skalierbare Anwendungsentwicklung. Darüber hinaus ist die Dokumentation sehr ausführlich und in mehreren Sprachen verfügbar.

### **5.2.1.2 Nachteile**

Die nicht zur Verfügung gestellten Komponenten stellen den größten Nachteil in diesem Vergleichsszenario dar. Für notwendige Komponenten, wie zum Beispiel Eingabemaske, Schaltfläche und Listenelement, bedarf es extra Planungs- und Entwicklungszeit. Die Gesamtentwicklungszeit sollte dementsprechend großzügig ausgelegt werden.

## 5.2.2 HTML5 und JavaScript Framework SproutCore

SproutCore (SC) ist ein 'open source'-Framework, das Programmierer beim Erstellen von Desktop-Web-Applikationen unterstützt und ausschließlich auf HTML5 und JavaScript setzt. Eine SC-Anwendung kann dadurch in allen aktuellen Browsern ausgeführt werden, besticht durch eine geringe Ladezeit und unmittelbare Reaktion auf Nutzeraktionen. Das Framework ist durch Apples Programmierschnittstelle Cocoa<sup>23</sup> inspiriert und implementiert das MVC-Architektur-Muster vollständig.

Der Erfinder Charles Jolley begann 2006, als Teil seines gegründeten StartUp-Unternehmens 'SproutIt', mit der Entwicklung des SproutCore-Frameworks [jolley 1]. 2006 heuerte er bei Apple als JavaScript Entwickler an und konnte das Framework bei der Entwicklung von MobileMe<sup>24</sup> und iWork<sup>25</sup> einsetzen und weiterentwickeln. Nach getaner Arbeit verließ Jolley im Juni 2010 Apple und gründete die Firma 'Strobe Digital Publishing'<sup>26</sup> (Strobe), um sich ausnahmslos der Weiterentwicklung von SC zu widmen. Strobe beschäftigt sich ausschließlich mit der Entwicklung von Rich-Web-Applikationen auf HTML5-Basis und bietet anderen Unternehmen Dienste rund um SC an. Die aktuellste SproutCore-Version ist 1.6<sup>27</sup> und ist seit dem 09. Juni 2011 erhältlich.

Charles Jolleys Philosophie<sup>28</sup>, das Framework und zur Verfügung gestellte Werkzeuge, ausnahmslos als open source zur Verfügung zu stellen, erscheint auf den ersten Blick eigenartig, ist in Kombination mit der großen Community aber plausibel und sehr effektiv. Das Framework wird durch Strobe stetig weiterentwickelt. Im selben Moment wird es von vielen Programmierern eingesetzt und weiterentwickelt. Durch die starke

---

23 <http://www.cocoa-coding.de/wasistcocoa.html>

24 <http://www.apple.com/de/mobileme/>

25 <http://www.apple.com/de/iwork/>

26 <http://gigaom.com/2010/07/13/charles-jolley-sproutcore-strobe/>

27 <http://blog.sproutcore.com/sproutcore-1-6-released/>

28 <http://blog.sproutcore.com/the-next-revolution/>

Vernetzung zwischen SC und der Community können sinnvolle extern erzeugte und von Strobe überarbeitete Erweiterungen in eine nachfolgende Version aufgenommen werden. Dies gewährleistet, dass sich die Kernausrüstung des Frameworks in die Richtung entwickelt die im Internet benötigt wird.

Die Verwendung des SC-Frameworks führt dazu, dass die Anwendungslogik einer Web-Anwendung vom Server auf den Client verlagert wird. Die anwendungsspezifischen Daten bleiben nach wie vor ausgelagert auf einem Server. Das Laden und die Verarbeitung der Daten übernimmt das SC-Framework in Kombination mit der vom Programmierer, mittels Data-Binding, LocalStorage, UserDefaults und anwendungsspezifischer Abläufe, erzeugten Logik. Dadurch werden Daten nur an den Server gesendet, wenn diese in der Datenbank aktualisiert werden müssen. Daraus resultiert, dass der Server im Vergleich zu herkömmlichen Webseiten zum Ausliefern und Empfangen von Datenpaketen benötigt wird und keine umfangreichen Verarbeitungsschritte abuarbeiten hat. Zum einen wird dadurch der Server entlastet und kann auf schnelle und zuverlässige Datenbereitstellung optimiert werden. Zum anderen wird durch weniger Datentransport die Fehleranfälligkeit reduziert.

Zusammenfassend ist festzuhalten, dass es sich bei SC-Anwendungen nicht um Web- oder Desktop-Anwendungen, sondern um eine Mischung aus beiden handelt. Die positiven Eigenschaften beider Anwendungstypen werden in einem neuen Anwendungstyp zusammengefasst und ergeben ein umfassendes Framework, dass für die Zukunft gewappnet und darüber hinaus einfach zu erweitern ist. Dieser neue Anwendungstyp kann als Cloud-Web-Anwendung, oder einfacher Cloud-Anwendung, bezeichnet werden. Der Begriff ist offiziell noch nirgends festgelegt worden, aber die SC-Entwickler nutzen ihn zur Erläuterung des Frameworks. Eine Cloud-Anwendung bietet vielseitiges Anwendungserlebnis, unmittelbare Reaktion des Systems auf

Nutzeraktionen, offline-Unterstützung, Plattformunabhängigkeit und Nutzbarkeit ohne Installation, vorausgesetzt ein moderner Browser und Internetzugang sind vorhanden. Eine solche Anwendung kann, wenn die zur Verfügung stehenden Techniken ausgeschöpft werden, als vollwertiges Programm betrachtet werden. [jolley 2]

### **5.2.2.1 Funktionsumfang**

Das SC-Framework stellt eine Vielzahl von Klassen und Programmiererleichterungen zur Verfügung, von denen in diesem Abschnitt einige aufgezählt und erklärt werden. Grundlegend ist festzuhalten, dass jede SC-Klasse von *SC.Object* abgeleitet ist und dadurch überall die selben Regeln gelten.

Eine Ableitung erfolgt, indem den Funktionen 'create()', 'design()' oder 'extend()', der Basisklasse, ein Objekt (data hash) übergeben wird. Dadurch wird die Klasse erweitert, zurückgegeben und auf einer Variable abgelegt. Die Funktion 'create()' wird benutzt, wenn von einer Klasse eine Instanz erzeugt werden soll. Zum erweitern einer Klasse wird 'extend()' benutzt und finden am häufigsten Verwendung. Die Funktion 'design()' entspricht 'extend()' und wird nur im 'resources' Verzeichnis verwendet.

Eine weitere Möglichkeit eine Klasse um Funktionalität zu erweitern bieten *Mixins*, diese werden einfach zusätzlich zu dem 'Object' (data hash) der Ableitungsfunktion übergeben. Als Beispiel ist das Mixin '*SC.ContentDisplay*' zu nennen, das Attribute zur Verfügung stellt, um dem Renderer mitzuteilen, welche Record-Attribute gerendert werden sollen.

DataBinding ist der Oberbegriff für verschiedenste Verbindungen, die zwischen einer View und einem Controller aufgebaut werden können. Verknüpfbare Werte sind Zahlen, Zeichenketten, Objekte und Boolean-Variablen. Mit Letzteren ist es möglich View-Elemente einfach zu deaktivieren oder auszublenden, dabei sind die Werte *YES* und *NO* genauso zu verwenden wie *true* und *false*.

Ein Binding kann auf mehrere Arten gesetzt werden, die Einfachste ist folgende:

```
valueBinding: 'MyApp.appController.text'
```

dieses Binding kann aber auch wie folgt erzeugt werden:

```
valueBinding: SC.Binding.from( 'MyApp.appController.text' ),
```

Ist unklar, ob ein *value* definitiv ein Boolean-Datentyp zurückgibt, kann der Rückgabewert transformiert werden, indem der Zusatz *!.bool()* angehängt wird:

```
valueBinding: SC.Binding.from( 'MyApp.appController.text' ).bool()
```

Dadurch wird garantiert, dass auf der View-Seite ein Boolean-Datentyp auftaucht.

Um eine Boolean-Abhängigkeit zu erzeugen, wird an die entsprechende View-Eigenschaft das Wort 'Binding' angehängt, wodurch z.B. folgende Bindings entstehen: *isVisibleBinding*, *isEditableBinding* und *isEnabledBinding*.

Dabei ist darauf zu achten, dass die entsprechende Variable oder Funktion (computed Property) des Controllers Boolean-Werte zurückgibt, denn diese verbundenen View-Eigenschaften akzeptieren nur den Boolean-Datentyp.

Eine Binding-Verbindung ist standardmäßig bidirektional, das bedeutet eine Veränderung auf der einen Seite wirkt sich unmittelbar auf die andere Seite aus. Es kann erforderlich sein, dass sich Werteänderungen nur in eine Richtung auswirken. Dazu stellt SC in der Klasse *SC.Bindings* mehrere Möglichkeiten bereit.

Als ebenfalls äußerst hilfreich erweisen sich die Techniken KVC (*Key-Value-Coding*) und KVO (*Key-Value-Observing*) für die Implementation von Funktionalität in Klassen. Bei KVC handelt es sich um Funktionen zum Setzen *'object.set( 'key', value )'* und

Abfragen `'object.get( 'key' )'` von Attributen, wodurch entsprechender Programmcode entfällt und Klassen, hauptsächlich Controller, übersichtlich bleiben.

Um Variablen (Attribute) zu überwachen stellt SC die Technik KVO (*Key-Value-Observing*) zur Verfügung. Funktionen können als *'computed Property'* (berechnete Eigenschaft) deklariert werden und stehen dadurch ebenfalls für *DataBinding* zu Verfügung. Die zweite Möglichkeit besteht darin, eine Funktion mit dem *observes( 'key1', ... )* Attribut zu erweitern und Änderungen eigener oder Eigenschaften anderer Controller zu überwachen. An dieser Stelle sollte erwähnt werden, dass alle Funktionen in SC „chainable“ sind, was bedeutet, dass verschiedene Funktionalitäten durch einen Punkt verkettet werden können.

SC setzt als Datenbasis das Datenaustauschformat JSON<sup>29</sup> ein. Es baut auf einer vergleichbaren Objektstruktur auf wie SC selbst und eignet sich daher hervorragend für den Datenaustausch mit einem Server und die interne Verarbeitung mit *SC.Record*-Hilfsobjekten.

Mit *Fixtures* können für ein Model sehr einfach Standarddaten definiert werden, um auch ohne Serveranbindung die Anwendung testen zu können.

Mit *SC.UserDefaults* können anwendungsspezifische Daten auf dem Client-System gespeichert und zu einem späteren Zeitpunkt ausgelesen werden.

Für die Arbeit mit Listen wird von SC die einheitliche Schnittstelle *SC.Enumerables* bereit gestellt. Sie stellt Funktionen zum Filtern von Record-Listen bereit und führt dadurch zu schlankem Programmcode.

---

<sup>29</sup> <http://www.json.org/json-de.html>

Im Zusammenhang mit Ereignissen, die von einer View erzeugt werden können und damit verbundenen Datenaktualisierungen, ist zu beachten, dass SC auf dem Programmkonstrukt *'Run Loop'*<sup>30</sup> (Ereignisschleife) basiert. Dabei werden Variablenänderungen erst nach Beendigung eines Loops aktiv, was unter Umständen zu Fehlverhalten der Anwendung führen kann. Vorbeugend kann ein *'Run Loop'* per Skript gestartet *'SC.RunLoop.begin()'* und beendet *'SC.RunLoop.end()'* werden. Zwischen den Anweisungen platzierte Aktualisierungen stehen im Anschluss unmittelbar zur Verfügung.

Zum Testen von implementierter Funktionalität bietet SC *'Unit Test'* an. Wird z.B. ein Controller mit dem Build-Tool *'sc-gen'* erzeugt, wird automatisch im Verzeichnis *'share\_it\tests\controllers\'* eine entsprechende Datei *'name\_test.js'* angelegt, diese kann um Funktionen erweitert werden, die bei der Ausführung einzelne Tests durchführen.

Das MVC-Entwurfsmuster wird in SproutCore um SDR (Server Interface, Display, Responder) erweitert und wie folgt implementiert:

In *'Models'* wird ein Teil der Anwendungslogik implementiert, indem Klassen mit entsprechenden Datentypen definiert werden. Verschiedene Klassen können Abhängigkeiten zueinander aufweisen, dies wird als Schema bezeichnet.

Das Schema kann mittels *'Server Interface'*, das eine Kombination von *SC.Store* und *SC.DataSource* darstellt, mit einer Datenbank verbunden werden. Der Store bildet dabei eine Art „lokale“ Datenbank, in die beim Start der Anwendung die „online“ Datensätze geladen werden. Diese können anschließend bearbeitet und mit der „online“ Datenbank synchronisiert werden.

---

<sup>30</sup> <http://blog.sproutcore.com/the-run-loop-part-1/>

## *5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung*

---

'Views' stellen in Kombination mit der 'Display'-Schicht den visuellen Teil der Anwendung dar und wandeln Maus-, Touch- und Keyboard-Ereignisse in anwendungsspezifische Ereignisse um. Die Display-Schicht enthält DOM-Bibliotheken, zum Beispiel jQuery oder Prototype, mit deren Hilfe HTML gerendert und das Ergebnis anschließen an den Browser, übergeben wird. View-Elemente werden mittels *DataBinding* mit Variablen eines Controllers verknüpft. Weiterführende Logik sollte in einer View nicht platziert werden.

Die '*Controller*' transportieren die Anwendungsdaten zwischen 'Models' und 'Views'. Sie reagieren auf Ereignisse, die zum Beispiel von einer View gesendet werden, um 'Models' bzw. 'Views', mittels *DataBinding*, auf dem aktuellsten Stand zu halten. Des Weiteren kann ein Controller als *Delegate* (Verantwortlicher) für eine Liste von Elementen erstellt werden. Dadurch ist es möglich auf Selektion und Drag&Drop -Ereignisse zu reagieren.

### **5.2.2.2 Vorteile**

Der Große Vorteil von SC ist, dass diverse Funktionalitäten und Elemente zur Verfügung gestellt werden, deren Entwicklung erheblichem Mehraufwand bedeuten würden. Der Programmierer kann sich fast ausschließlich auf die Implementation der Anwendungslogik konzentrieren und muss nicht erst diverse Funktionalitäten selbständig in Klassen kapseln. Darüber hinaus werden von Strobe regelmäßig neue Funktionen veröffentlicht um die Entwicklungsmöglichkeiten zu verbessert.

Ein weiterer Vorteil ist, dass SC auf HTML5 und JavaScript setzt und dadurch in allen neuen Browsern eingesetzt werden kann. Da der neue, noch nicht fertiggestellte, Seitenbeschreibungsstandard in vielen Bereichen ebenfalls auf JavaScript setzt, bietet SC eine umfangreiche Basis um ansprechende Web-Auftritte zu realisieren.

### **5.2.2.3 Nachteile**

Als Nachteile von SproutCore können JavaScript und die fehlende IDE genannt werden. Surft ein Nutzer mit deaktiviertem JavaScript auf eine Webseite die mit SC erstellt wurde bekommt er im schlimmsten Fall eine leere Seite angezeigt. In diesen Fall könnte ein entsprechend aussagekräftiger Hinweis für Abhilfe sorgen.

Die fehlende IDE stellt auf den ersten Blick einen großen Nachteil dar. Ein Entwickler, der es gewohnt ist damit zu arbeiten, wird die fehlenden Funktionen und Hilfsmittel als sehr großen Nachteil empfinden, da es für ihn normal ist Code teilweise generiert und Attribute und Funktionen/Methoden einer Klasseninstanz mit zusätzlichen Informationen in einem Kontextmenü angezeigt zu bekommen.

Das es für das SC-Framework noch keine IDE<sup>31</sup> gibt, liegt wahrscheinlich an dem noch jungen Entwicklungsstand. Die Dokumentation des Framework ist lückenhaft und für diverse Anwendungsszenarien fehlen informative Beispiele, aber die Entwickler von Strobe und die Community arbeiten hart daran diese Missstände umzukehren.

## **5.3 Verwendete Entwicklungsumgebungen**

In der Softwareentwicklung kommen spezielle Hilfsprogramme zum Einsatz. Dabei kann von einem einfachen Editor, der kaum Funktionalität bietet, bis zu einer mit Funktionen strotzenden IDE, alles eingesetzt werden. Web-Formate wie .html, .js, .php oder .as können mit einem einfachen Editor bearbeitet werden. In Flash wird zur Anordnung grafischer Elemente, Entwicklung umfangreicher Komponenten und finaler Erzeugung einer SWF (kompilieren), die Flash-IDE (aktuellste Version ist die CS5) oder externe Software mit einem integrierten Compiler benötigt.

### **5.3.1 Flash/PureMVC**

Als Entwicklungsumgebung wurde der Adobe Flash Builder 4, ehemals Adobe Flex Builder, gewählt. Es handelt sich dabei um ein auf der Eclipse Plattform<sup>32</sup> aufbauendes Werkzeug zur schnellen Entwicklung plattformübergreifender und reichhaltiger Web-Applikationen. Der Flash Builder bietet unter anderem einen integrierten Code-Editor mit einstellbarer Syntax-Hervorhebung und Code-Hinweisen, einen Debugger mit dem zur Laufzeit, Schritt für Schritt, durch die einzelnen Programmstufen navigiert werden

---

31 [http://de.wikipedia.org/wiki/Integrierte\\_Entwicklungsumgebung](http://de.wikipedia.org/wiki/Integrierte_Entwicklungsumgebung)

32 [http://de.wikipedia.org/wiki/Eclipse\\_\(IDE\)](http://de.wikipedia.org/wiki/Eclipse_(IDE))

kann, um Variablen auszulesen oder Logikfehler aufzuspüren. Der eingebaute Datei-Explorer bietet Überblick über alle dem Projekt zugehörigen Klassen und Methoden und stellt das zentrale Navigationselement dar. Des Weiteren können diverse Zusatzfunktionalitäten hinzugefügt werden, z.B. SVN<sup>33</sup> das die gemeinsame Arbeit an einem Projekt vereinfacht.

### **5.3.1.1 Vorteile**

Der Flash Builder 4 bietet einem Programmierer alles was für die Entwicklung komplexer Anwendungen notwendig und hilfreich ist. Er verfügt über Intellisense für Autovervollständigen ActionScript spezifischer oder eigener Funktionen. Für Instanzen von Klassen werden, nach dem Punkt-Operator, die verfügbaren public Variablen und Methoden mit Zusatzinformationen in einer Dropdown-Liste dargestellt. Darüber hinaus kann mit gedrückter 'Strg'-Taste und 'linker'-Maustaste direkt eine Funktion einer Klasse aufgerufen werden. Die entsprechende ActionScript-Datei wird geöffnet und an die entsprechende Stelle gesprungen.

Ein ebenfalls sehr hilfreiches Werkzeug ist der Debugger. Mit ihm lassen sich selbst in sehr komplexen Anwendungen Variablen und Abläufe zur Laufzeit überwachen, auswerten und gegebenenfalls ineffizienter Code beseitigen. Des Weiteren besteht die Möglichkeit weitere Module mit Zusatzfunktionen hinzuzufügen.

---

<sup>33</sup> [http://de.wikipedia.org/wiki/Apache\\_Subversion](http://de.wikipedia.org/wiki/Apache_Subversion)

### **5.3.1.2 Nachteile**

Der Flash Builder 4 ist ein umfangreiches und komplexes Werkzeug zum Erstellen verschiedenster Anwendungen. Als Nachteil in Bezug auf die Entwicklung, der in dem Vergleichsszenario zu erstellenden Anwendung „ShareIt“, kann nur der Kostenfaktor genannt werden.

### 5.3.2 SproutCore

Für die Entwicklung der SproutCore „ShareIt“ Anwendung wird der frei verfügbare Notepad++ Editor<sup>34</sup> mit dem zusätzlichen 'Explorer' PlugIn<sup>35</sup> verwendet. Er kennt die Syntax der gängigsten Programmier- und Auszeichnungssprachen und hebt dadurch Schlüsselwörter farbig hervor, wodurch der Programmcode Übersichtlicher dargestellt wird. Besonders hilfreich sind die Multi-Ansichten, dabei lassen sich mehrere Dateien als Tabs nebeneinander öffnen, wodurch z.B. einfach zwischen View und Controller hin und her gewechselt werden kann. Die geöffneten Dateien werden in einer Sitzung gespeichert und stehen dadurch nach Schließen und Öffnen des Editors wieder zur Verfügung. Die 'Suchen und Ersetzen'- Funktion des Explorer-PlugIn`s beschränkt sich nicht nur auf eine Datei, sondern kann auf ganze Verzeichnisbäume angewendet werden. Dadurch ist es zum Beispiel sehr einfach einen Controller umzubenennen.

Der in der Entwicklung befindliche 'View'-Builder 'Greenhouse'<sup>36</sup> wurde für diesen Vergleich nicht eingesetzt. Zum einen gibt es Kompatibilitätsprobleme mit unterschiedlichen SC-Versionen und zum anderen könnte dadurch keine detaillierte Erläuterung dieses Akteurs erfolgen.

Das SC-Framework stellt umfangreiche 'build-tools' zur Verfügung, mit denen z.B. ein neuen Controller erzeugt werden kann. Dabei wird die js.-Datei im dafür vorgesehenen Verzeichnis angelegt und mit einem Basis-Code ausgestattet. Darüber hinaus wird eine entsprechende 'unit-test'-Datei im Verzeichnis '...\test\controllers\' angelegt.

Die SC-'build-tools' sind in Ruby<sup>37</sup> entwickelt und werden daher über die Ruby-

---

34 <http://notepad-plus-plus.org/download>

35 <http://sourceforge.net/projects/npp-plugins/files/Explorer/>

36 <http://www.golem.de/1004/74628.html>

37 <http://www.ruby-lang.org/de/>

Kommandozeile verwendet. Eine ausführliche Anleitung<sup>38</sup> zur Installation von Ruby und dem SC-Framework finden sich auf der SproutCore-Webseite.

### **5.3.2.1 Vorteile**

Als Vorteil kann ganz klar hervorgehoben werden, dass der Notepad++ Editor und diverse Erweiterungen frei zur Verfügung steht. Dadurch ist die Entwicklung einer SC-Anwendung komplett ohne Anschaffungskosten realisierbar. Des Weiteren ist der Programmierer nicht an diesen Editor gebunden, nahezu jeder Editor kann verwendet werden.

### **5.3.2.2 Nachteile**

Notepad++ verfügt über keine SproutCore spezifischen JavaScript-Kenntnisse. Dadurch kann er keine Wortvervollständigung oder Variablen und Methoden eines Objektes anzeigen. Ein weiterer Nachteil ist ein fehlender Debugger, dies kann durch strukturierte Programmierung, den Einsatz von Unit-Tests und der Nutzung der JavaScript-Console (Browser) vernachlässigt werden.

---

<sup>38</sup> [http://www.sproutcore.com/install\\_win/](http://www.sproutcore.com/install_win/) für Windows

### **5.3.3 Schlussfolgerung**

Die in Kapitel 5.3 aufgezählten Vor- und Nachteile der zwei verwendeten Entwicklungsumgebungen sprechen aus Sicht einer nicht kommerziellen Weiterführung der Anwendung „ShareIt“ klar für SproutCore und HTML5. Das SC-Framework, die verwendeten Entwicklungswerkzeuge und die eingesetzte Datenbank (CouchDB<sup>39</sup>) sind kostenlos, wodurch mit minimalen Mitteln ein Projekt realisierbar ist. Die Nachteile der gewählten Entwicklungswerkzeuge sind für einen erfahrenen Programmierer durch strukturierte und überlegte Programmierung zu vernachlässigen. Darüber hinaus relativieren die Nullkosten diese fehlenden Funktionen schnell.

---

<sup>39</sup> <http://www.couchbase.com/downloads>

## 5.4 Framework Einstieg/Basisklassen

Die beiden Frameworks ermöglichen auf unterschiedliche Weise den Einstieg in die Projektentwicklung, diese werden im Folgenden erläutert.

### 5.4.1 Flash/PureMVC spezifisch

Nach dem Erzeugen eines neuen ActionScript-Projekts, mit dem Name 'ShareIt', stellt die Klasse 'ShareIt.as' im Paket 'flash' den unmittelbarer Einstiegspunkt in die Anwendungsentwicklung dar. Sie initialisiert die konkrete Facade 'ShareItFacade' und fügt Container für sämtliche Content-Elemente hinzu. Die konkrete Facade registriert im Zuge der Initialisierung in der Funktion '*initializeCommands()*' alle relevanten Commands mit der Funktion '*registerCommand()*' und in '*initializeModel()*' den für die Ansichtswechsel zuständigen 'ProxyScreenManager'.

Im Anschluss wird die Anwendung gestartet, indem die Funktion 'startUp()' der 'ShareItFacade' aufgerufen und dadurch das 'CommandStartup' ausgeführt wird.

Als eines der wichtigsten Elemente muss der 'DataRuntimeLoader' hervorgehoben werden. Ihm lassen sich Instanzen von 'LoadingEntry' übergeben und er verwaltet die individuellen Ladevorgänge. Einem 'LoadingEntry' können Events hinzugefügt werden, deren definierte Callback-Funktion wird bei Eintreten des Events aufgerufen und eine Reaktion kann erfolgen.

### 5.4.2 SproutCore spezifisch

Nachdem über die Ruby-Kommandozeile mit `'sc-init shareit'` das Projekt erzeugt wurde erfolgt der Einstieg mit dem Verzeichnisnamen der Anwendung, in diesem Fall `'share_it'`, in dem sich ein Unterverzeichnis `'apps'` und eine Datei `'Buildfile'` befindet.

In der Datei `'Buildfile'` werden im Fall von „ShareIt“ mit zwei Anweisungen hinterlegt.

```
proxy '/shareit_shareobjects', :to => 'localhost:5984'
```

```
proxy '/shareit_persons', :to => 'localhost:5984'
```

Damit werden zwei Umleitungen definiert, die für die Verbindung mit der externen CouchDB zuständig sind.

Das Verzeichnis `'apps'` beinhalten ein weiteres Verzeichnis mit dem Anwendungsnamen `'share_it'`, in dem sich die relevanten Projektdateien in Unterverzeichnissen, die die drei MVC-Akteure widerspiegeln, befinden. Dieses Unterverzeichnis `'\share_it\apps\share_it\...'` wird im Folgenden als Ausgangsverzeichnis `'share_it'` für Pfadangaben verwendet.

Das soeben definierte Ausgangsverzeichnis beinhaltet noch zwei weitere Dateien, die für die Initialisierung und den Start der Anwendung zuständig sind.

In `'core.js'` wird mittels Vererbung eine Instanz von `SC.Application` erzeugt und auf der Variable `'ShareIt'` gespeichert. Die Application erhält einen `NAMESPACE`, über den auf alle Instanzen zugegriffen werden kann, und eine `VERSION`, die die verwendete SC-Version definiert. Darüber hinaus werden zwei `SC.Store` Instanzen erzeugt und mit einem `DataSource` verknüpft.

Die Datei `'main.js'` definiert eine `'main()'`-Funktion die die `'ShareIt.main()'` aufruft und

die Anwendung startet. Die 'ShareIt.main()' registriert relevante Routen (URL-Varianten) und veranlasst den 'ShareIt.personStore' zum Laden der Datensätze vom Server, damit ein Anmeldevorgang durchgeführt werden kann.

## **5.5 Implementierung der Models**

Die Anwendung „ShareIt“ benötigt für die relevanten Daten zwei verschiedene Datenmodelle.

Zu verwaltende Tauschobjekte werden mit 'ShareObject' abgebildet. Dieses *Model* bildet die Informationen eines Objektes ab und verfügt über Attribute, die eine eindeutige Zuordnung zu einem Eigentümer, aktuellen Besitzer und beteiligten Personen zulässt.

Personen, die sich registrieren, werden mit dem *Model* 'Person' abgebildet. Alle notwendigen Attribute, die zur Kontaktierung einer Person notwendig sind, werden angelegt. Aus Sicht der Person gibt es kein Attribut, das eine Verknüpfung zu einem oder mehreren Tauschobjekten verweist.

### **5.5.1 PureMVC Models & Proxies**

Ein *Model* wird als AS-Klasse in dem Paket '*shareit.model.data*' angelegt und erbt von der Klasse '*Object*'. Die erforderlichen Eigenschaften werden als *public*-Attribute deklariert, wodurch der direkte Zugriff ermöglicht wird.

Das *Model* 'DataShareObject' definiert alle für ein Tauschobjekt notwendigen Daten und

---

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

initialisiert diese mit einem Standardwert. Die Attribute 'owner' und 'currentOwner' sind vom Datentyp Object, weil dieser beliebige Daten aufnehmen kann. Aus Gründen der Entwicklungszeit wurde an dieser Stelle eine flexible Typisierung gewählt.

Das Attribut 'participants' ist vom Datentyp Array und fungiert als Liste der Personen, die an einem Tauschobjekt beteiligt sind.

---

### **shareit.model.data.shareobject.DataShareObject**

---

```
public class DataShareObject extends Object {
    public var id:String = "";
    public var title:String = "";
    public var description:String = "";
    public var owner:Object = {};
    public var currentOwner:Object = {};
    public var participants:Array = new Array();
    public var history:Array = new Array();
    public var defects:Array = new Array();
    public function DataShareObject() {
        super();
    }
}
```

---

Legt der Nutzer ein neues Tauschobjekt an, wird ein neuer 'ProxyShareObject' erzeugt und bekommt eine neue Instanz dieses Modells übergeben. Die Model-Instanz wird auf dem Attribut 'data' des Proxys gespeichert und anschließend mit Werten gefüllt.

Das *Model* 'DataUser' definiert alle Eigenschaften, die für die eindeutige Identifizierung und Kontaktaufnahme notwendig sind. Aus Gründen der Komplexität werden die Eigenschaften Name, Straße und Adresse nicht als einzelne Attribute, sondern mit dem Attribut 'data' vom Typ XML definiert. Zur Laufzeit wird darauf ein XML-Knoten mit

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

den Tags forename, surname, street, number, city, zipcode, phone und email gespeichert.

---

### **shareit.model.data.DataUser**

---

```
public class DataUser extends Object {
    public var ID:String = "";
    public var data:XML;
    public var ownShareObjects:Array = new Array();
    public var participations:Array = new Array();
    public function DataUser() {
        super();
    }
}
```

---

Zur Laufzeit wird dieses Model als Instanz dem 'ProxyUserData' übergeben und auf dem Attribut 'data' (Object) gespeichert.

Wie soeben beschrieben, werden die Models auf das Attribut 'data', vom Datentyp Object, abgelegt. Um auf die Attribute des jeweiligen Models Zugriff zu erhalten, muss 'data' auf den entsprechenden Datentyp (*Model*) gecastet werden. Im 'ProxyUserData' wird dazu die *getter*-Funktion 'dataUser' benutzt:

---

### **shareit.model.proxy.userdata.ProxyUserData - Ausschnitt**

---

```
protected function get dataUser():DataUser {
    return data as DataUser;
}
```

---

In jedem Proxy gibt es vergleichbare Funktionen, die die Datentypen sicherstellen.

Nach erfolgreichem Login übergibt 'ProxyLogin' die vom Server erhaltenen Nutzerdaten (XML) mit der Funktion 'setUserData()' an den 'ProxyUserData'.

---

### **shareit.model.proxy.userdata.ProxyUserData**

---

```
public function setUserData( value:XML ):void {
    dataUser.data = value;
    dataUser.ID = dataUser.data.@id.toString();
}
```

---

Diese Funktion nutzt die zuvor vorgestellte *getter*-Funktion um Zugriff auf die Attribute der gespeicherten Model-Instanz zu erhalten und speichert die Daten.

Der Austausch von Daten zwischen Anwendung und Server kann sehr gut am 'ProxyLogin' erläutert werden. Dieser Proxy wird zum Anwendungsstart registriert und bekommt, wenn die Nutzereingaben vollständig und korrekt sind, vom 'MediatorScreenLogin' an die Funktion 'sendData()' einen XML-Knoten mit den Logindaten übergeben. In Folge dessen werden die Daten an ein Server-Skript gesendet und Events für eine Serverantwort registriert.

---

### **shareit.model.proxy.ProxyLogin - reduziert**

---

```
public function sendData( inputXml:XML ):void {
    _loadingEntry = new LoadingEntry();
    _loadingEntry.type = "data";
    _loadingEntry.vars = new URLVariables();
    _loadingEntry.vars.session = ShareIt.SESSION_ID;
    _loadingEntry.vars.userdata = inputXml;
    _loadingEntry.url = proxyApplication.getService("login").url;
    _loadingEntry.method = proxyApplication.getService("login").method;
    _loadingEntry.addEventListener(LOADING_ERROR, onLoginSendError);
    _loadingEntry.addEventListener(LOADING_COMPLETE, onLoginSent);
    DataRuntimeLoader.load(_loadingEntry);
}
```

---

Im ersten Schritt wird eine neue Instanz von 'LoadingEntry' erzeugt und auf der Proxy-Variablen '\_loadingEntry' gespeichert. Im zweiten Schritt werden Attribute den Remote-Vorgang betreffend mit Werten gefüllt und im dritten Schritt wird der Funktion 'load()' des 'DataRuntimeLoader' die Instanz '\_loadingEntry' übergeben. Sind die Daten vollständig übertragen, wird vom Server geprüft, ob die Logindaten mit einem Datensatz übereinstimmen, eine entsprechende Antwort generiert und zurückgesendet. Mit dem Event 'LoaderEvent.LOADING\_COMPLETE' und der definierten Funktion 'onLoginSent()' wird diese Antwort entgegen genommen.

---

### **shareit.model.proxy.ProxyLogin - reduziert**

---

```
protected function onLoginSent( e:LoaderEvent ):void {
    setData( XML( e.data ) );
    if( data.result.toString() == "success" ) {
        proxyApplication.setSessionID( data.session_id.toString() );
        if( data.hasOwnProperty( "userdata" ) )
            proxyUserData.setUserData( XML( data.userdata ) );
        if( data.hasOwnProperty( "shareobjects" ) )
            proxyUserData.setOwnShareObjets( XML( data.shareobjects ) );
        sendNotification( Notify.LOGIN_SUCCESS );
    } else if( data.result.toString() == "fault" ) {
        sendNotification(Notify.LOGIN_FAULT, data.userinfo.toString());
    } else
        sendNotification( Notify.LOGIN_ERROR, "Beim übertragen..." );
}
```

---

Die empfangenen Daten werden als erstes mit der Funktion 'setData()', die von der Basisklasse 'Proxy' geerbt wird, auf das Attribut 'data' gespeichert. Im Anschluss wird die Serverantwort überprüft und entsprechend reagiert. Bei keiner Übereinstimmung

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

wird die *Notification* 'LOGIN\_FAULT' versendet, diese wird vom 'MediatorScreenLogin' empfangen und eine entsprechende Meldung angezeigt. Ist der Login korrekt, wird der Datensatz aus der Serverantwort ausgelesen und der Funktion 'setUserData()' des 'ProxyUserData' übergeben. Anschließend wird die *Notification* 'LOGIN\_SUCCESS' versendet, auf die ebenfalls der 'MediatorScreenLogin' reagiert.

Proxies sind nicht nur für die Datenhaltung und Kommunikation mit Remote-Services zuständig, sie übernehmen auch Teilaufgaben von Commands.

Im konkreten Fall wird ein Proxy benutzt, um den Wechsel von aktueller zu neuer Ansicht durchzuführen. Der 'ProxyScreenManager' erhält vom 'CommandSwitchScreen' über die Funktion 'showScreen()' eine Referenz des neu anzuzeigenden Mediators.

---

### **shareit.model.proxy.ProxyScreenManager - reduziert**

---

```
public function showScreen( screen:MediatorScreen ):void {  
    ...  
    var currentMediatorScreen:MediatorScreen =  
        facade.retrieveMediator( "Mediator" +  
            dataScreenManager.currentScreenName ) as MediatorScreen;  
    if( currentMediatorScreen ) {  
        facade.removeMediator( currentMediatorScreen.getMediatorName() );  
        currentMediatorScreen.screen.mouseEnabled = false;  
        currentMediatorScreen.screen.hide();  
    }  
    ...  
    var newScreen:Screen = screen.getViewComponent() as Screen;  
    ...  
    newScreen.show();  
    facade.registerMediator( screen );  
}
```

---

Die Funktion entfernt mit `'facade.removeMediator()'` den Mediator der aktuellen View aus dem System, deaktiviert und blendet die aktuelle View aus. Im Anschluss blendet sie die neue View ein und registriert den Mediator mit `'facade.registerMediator()'` am System. Auf vergleichbare Weise ist der `'ProxyScreenManager'` für das anzeigen und ausblenden von Popups zuständig.

Der `'ProxyDefault'` implementiert zwei *getter*-Funktionen (`proxyApplication` und `proxyUserData`), die die am häufigsten verwendeten Proxies darstellen. Alle anderen Proxies erben von diesem Proxy und erhalten dadurch direkten Zugriff.

Der `'ProxyApplication'` ist als *getter*-Funktion in Proxies, Mediatoren und Commands aufgrund der Vererbung zugänglich und stellt Zugriff auf Basisdaten und -funktionen zur Verfügung.

Der `'ProxyPreloader'` wird beim Anwendungsstart registriert und ausgeführt. Er lädt im ersten Schritt eine `'config.xml'`, diese enthält Basisdaten und in einem `'preload'`-Knoten weitere Daten, die geladen werden müssen, bevor die Anwendung verwendet werden kann.

Der `'ProxyRegister'` bekommt von dem `'MediatorScreenRegister'` Nutzereingaben in Form einer XML übergeben und sendet diese an ein Server-Skript.

## 5.5.2 SproutCore Models

Jedes Model wird in SC durch Ableitung von *SC.Record* erzeugt. Standardmäßig nutzt SC *'guid'* als key-Attribut, dies kann in *'primaryKey'* umbenannt werden. Unabhängig von dem tatsächlich benutzten key-Attribut kann dieses mit *record.get('id')* abgefragt werden. Weitere Attribute können mit *'attribute helper'*-Objekten, speziellen Helferobjekten, die als *'computed Property'* agieren, angelegt werden. SC stellt für alle JavaScript Grunddatentypen diese Helferobjekte zur Verfügung. Zusätzlich gibt es ein spezielles Objekt (*SC.DateTime*) für Datum- und Zeitinhalte.

Ein Helferobjekt kann mit zusätzlichen Optionen (*defaultValue*, *isRequired*) ausgestattet werden. Ein mit diesen Optionen ausgestattetes Attribut sieht wie folgt aus:

---

**Beispiel: Attribut 'name' mit Standardwert**

---

```
name: SC.Record.attr( String, {
  isRequired: YES,
  defaultValue: 'nobody' } ),
```

---

Dadurch ist es möglich, einem Attribut *'name'* per *record.set( 'name', 123 )* einen beliebigen Wert, in diesem Fall eine Zahl, zuzuweisen, das Helferobjekt *'String'* wandelt den Wert automatisch in den String *'123'* um.

Alle Attribute eines *'Models'* sollten mit *'attribute helpers'* ausgestattet werden. Dadurch wird die Konsistenz der Daten sichergestellt und unerwünschte Effekte zur Laufzeit vermieden.

Das *Model* *'ShareIt.ShareObject'* definiert alle für ein Tauschobjekt notwendigen Daten. Spezielle Bedeutung besitzen die Attribute *'mainOwner'*, *'currentOwner'* und *'persons'*,

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

sie sind ausschlaggebend für die Filterung der anzuzeigenden Tauschobjekte und Anzeige der Eigenschaften eines selektierten Tauschobjektes.

---

### share\_it\models\ShareObject.js - reduziert

---

```
ShareIt.ShareObject = SC.Record.extend({
  primaryKey:   "_id",
  objectId:     SC.Record.attr( String ),
  title:        SC.Record.attr( String, { isRequired: YES } ),
  description:  SC.Record.attr( String, { isRequired: YES } ),
  mainOwner:    SC.Record.attr( String, { isRequired: YES } ),
  currentOwner: SC.Record.attr( String, { isRequired: YES } ),
  persons:      SC.Record.attr( String, { isRequired: YES } ),
  image:        SC.Record.attr( String )
});
```

---

Das 'ShareIt.Person' Model beschreibt ausschließlich Daten eine Person betreffend. Alle Verweise auf Zugehörigkeiten werden in einem Tauschobjekt gespeichert. Des Weiteren enthält dieses Model zur Veranschaulichung eine '*computed Property*' (fullName), über die der komplette Name einer Person ermittelt werden kann.

---

### share\_it\models\Person.js - reduziert

---

```
ShareIt.Person = SC.Record.extend({
  primaryKey:   "_id",
  userid:       SC.Record.attr( String ),

  username:     SC.Record.attr( String, { isRequired: YES } ),
  password:     SC.Record.attr( String, { isRequired: YES } ),
  type:         SC.Record.attr( String, { isRequired: YES,
                                          defaultValue: 'user' } ),
  forename:     SC.Record.attr( String, { isRequired: YES,
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
                                defaultValue: 'Test' })),
surname:      SC.Record.attr( String, { isRequired:  YES,
                                defaultValue: 'Tester' })),
...
fullName:     function() {
    return this.getEach( 'forename', 'surname' ).compact().join(' ');
  }.property( 'forename', 'surname' ).cacheable()
});
```

---

Die erstellten Models bilden die Basis für die Verarbeitung von Datensätzen aus einer Datenquelle. Für diese Anwendung wurde eine nicht relationale Datenbank ausgewählt, weil dieses Konzept die Verlagerung der Anwendungslogik vom Server auf die Client-Seite auf einfachste Weise unterstützt.

In jedem Datensatz sind Daten in einem 'data hash' als Strings enthalten. Die in Models erzeugte Datenstruktur ermöglichen es SC automatisch diese Strings in Objekte umzuwandeln und somit im gewünschten *Model*-Datentyp abzuspeichern. Der Zugriff auf Record-Attribute ist dadurch, nach abgeschlossenem Ladevorgang, sofort möglich.

Die Verbindung zwischen *SC.Store* und dem, für diese Anwendung gewählte, relationale Datenbankkonzept *CouchDB* ist relativ unkompliziert zu integrieren.

In Anlehnung an die zwei erläuterten Models wurden auf dem CouchDB-Server zwei getrennte Datenbanken ('shareit\_persons', 'shareit\_shareobjects') angelegt.

Um einen *SC.Store* mit CouchDB, zu verbinden muss eine Sub-Klasse von *SC.DataSource* erzeugt und die zur Record-Verwaltung notwendigen Funktionen (createRecord, updateRecord und destroyRecord) überschrieben werden. Im Fall von 'Person' sieht dies wie folgt aus:

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

### **share\_it\data\_source\PersonDataSource.js - reduziert**

---

```
ShareIt.PersonDataSource = SC.DataSource.extend({
  _dbpath: 'shareit_persons',
  getServerPath: function( resourceName ) {
    var path = '/' + this._dbpath + "/" + resourceName;
    return path;
  },
  getServerData: function( dbName ) {
    var path = '/' + this._dbpath + "/_design/app/_view/" + dbName;
    return path;
  },
  fetch: function( store, query ) {
    SC.Request.getUrl(this.getServerData('allPersons')).json()
      .header( 'Accept', 'application/json' )
      .notify( this, 'didFetchPersonsData', store, query )
      .send();
    return YES; // return YES if you handled the query
  },
  didFetchPersonsData: function( response, store, query ) {
    var body = response.get( 'encodedBody' );
    var couchResponse = SC.json.decode( body );
    var records=couchResponse.rows.getEach( 'value' );
    store.loadRecords( ShareIt.Person, records );
    store.dataSourceDidFetchQuery( query);
  },
  createRecord: function( ...
  updateRecord: function( ...
  destroyRecord: function( ...
});
```

---

Durch den 'open source'-Charakter dieses Frameworks wird dem Programmierer, wie in diesem Fall von der Community, die notwendige Klasse<sup>40</sup> zur Verfügung gestellt. Nach geringen Anpassungen ist die Anwendung mit dem CouchDB-Server verbunden und lädt Daten aus der Datenbank.

Angepasst werden muss nur der Name der zu ladenden Datenbank

```
_dbpath: 'shareit_persons'
```

und das den zu ladenden Records zu Grunde liegende *Model*, in diesem Fall 'ShareIt.Person'.

Die Funktion '*fetch()*' wird vom *SC.Store* aufgerufen, wenn mit der Datenquelle synchronisiert werden soll. Dabei wird eine Anfrage an eine spezielle URL geschickt, die von der Funktion '*getServerData('allPersons')*' zurückgegeben wird. Der übergebene Parameter entspricht der Filter-Funktion der Datenbank (z.B. alle freigeschalteten Nutzer). Der komplette Pfad lautet im Fall von Person: '/shareit\_persons/\_design/app/\_view/allPersons'

Sind die Rohdaten (verschlüsselt vom CouchDB-Server versendet) vollständig geladen, wird die Funktion '*didFetchPersonsData()*' aufgerufen. Diese entschlüsselt die Rohdaten und speichert auf der lokalen Variable 'records' die einzelnen Datensätze (Array). Der Store matcht anschließend die Rohdaten aller Datensätze auf das *Model*. Dazu wird der Funktion '*loadRecords()*' das *Model* und das Array übergeben.

---

<sup>40</sup> <http://wiki.sproutcore.com/w/page/31341406/Todos%2006%20-%20Building%20with%20CouchDB>

Die Initialisierung eines *SC.Store* mit der erweiterten *SC.DataSource* erfolgt mit folgender Anweisung in der 'core.js' im Verzeichnis 'share\_it'.

---

**share\_it\core.js - Ausschnitt**

---

```
personStore: SC.Store.create({
  commitRecordsAutomatically: YES
}).from( 'ShareIt.PersonDataSource' )
```

---

Die Eigenschaft 'commitRecordsAutomatically' besagt, dass Änderungen an einem bestehenden Record automatisch mit der Datenbank synchronisiert werden. Wird diese Eigenschaft auf NO gesetzt, müsste zur Synchronisation eines Record die Funktion '*commitRecord()*' aufgerufen werden.

### 5.5.3 Schlussfolgerungen

Ein PureMVC *Model* wird als Instanz einer Klasse erzeugt und von einem Proxy befüllt. Dabei übernimmt der Proxy das Laden und zuweisen der Daten und muss diese entsprechend aufbereiten und die Datentyp-Sicherheit gewährleisten.

In SC stellt ein *Model* im einfacheren Sinn eine Datentyp-Vorlage dar. Die notwendigen Attribute werden mit Helferobjekten und evtl. Standardwerten angelegt. Einer *SC.Store* Instanz wird anschließend mittels dem Model mitgeteilt, wie die Datensätze (Records) zu interpretieren sind. Im Anschluss kann unmittelbar, mittels *get()* und *set()*, mit den Attributen der Records gearbeitet werden. Die Erweiterung des Models ist genauso denkbar einfach. Das Attribut wird hinzugefügt und kann verwendet werden. In PureMVC wären in einem solchen Fall etliche Zusatzschritte in Proxies notwendig, bevor das neue Attribut verwendet werden kann.

Der Programmierer kann sich unter Verwendung von SC auf das Wesentliche konzentrieren und muss nicht die interne Verwaltung der Daten implementieren. Dadurch wird die Basis der Anwendungslogik schmal gehalten und die Struktur nicht unnötig aufgebläht.

Im konkreten Fall ließ sich mit SC mit wenigen Anweisungen eine funktionierende und einfach zu erweiternde Datenhaltung programmieren. Die Helferobjekte, die die Korrektheit der Daten sicherstellen, sind logisch anzuwenden und mit wenig Handgriffen um Standardwerte erweitert.

## **5.6 Implementierung der Views**

Die Anwendung „ShareIt“ erfordert auf der einen Seite Views die Dateneingaben zulassen und auf der anderen Seite Views die Daten anzeigen.

Im ersten Schritt bekommt der Nutzer die *LoginView* angezeigt. Auf dieser kann er sich entweder einloggen oder, wenn er noch nicht registriert ist, die *RegisterView* aufrufen. Beide Views stellen gezielte Eingabemöglichkeiten zur Verfügung, übermitteln die Daten an den zuständigen Controller und zeigen gegebenenfalls eine Fehlermeldung an. Hat sich der Nutzer erfolgreich angemeldet erscheint die *IntroView*, die Informationen zur Anwendung und allen damit verbundenen Aktionen anzeigt. Ab diesem Punkt hat der Nutzer die Wahl zwischen folgenden Views. Er kann ein neues Tauschobjekt anlegen, sich alle im System befindlichen Tauschobjekte anzeigen, diese nach eigenen, beteiligten oder im eigenen Besitz befindlichen Tauschobjekten filtern und sich alle registrierten Nutzer auflisten lassen.

## 5.6.1 PureMVC Views & Mediators

Eine View wird mit verschiedenen Elementen ausgestattet, um den gewünschten Zweck zu erfüllen, dabei ist es von Vorteil, wenn die Komponenten mit kurzen Anweisen zu verwenden sind. Wichtige Komponenten sind in dieser Anwendung Textausgabe, Texteingabe und Schaltflächen. Der Aufbau und die Verwendung dieser Elemente (Klassen) wird in diesem Abschnitt erläutert.

### 5.6.1.1 View Basisklassen

Die Views heißen in der PureMVC „ShareIt“-Anwendung 'Screens' und sind von der Basisklasse 'Screen' abgeleitet. Diese Klasse erbt ihrerseits von der Klasse 'components.display.Sprite', die als Basis für fast alle visuellen Komponenten der Anwendung verwendet wird.

---

#### **components.display.Sprite - reduziert**

---

```
public dynamic class Sprite extends flash.display.Sprite {
    public function hide( e:Event = null ):void {
        TweenMax.to( this, hideDuration, {
            alpha: 0,
            ease: _hideEasing,
            onComplete: hideComplete } );
    }
    public function show( e:Event = null ):void {
        TweenMax.to( this, showDuration, {
            alpha: 1,
            ease: _showEasing,
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
        onComplete: showComplete } );  
    }  
    protected function hideComplete():void {  
        dispatchEvent( new SpriteEvent( HIDE_COMPLETE ) );  
    }  
    protected function showComplete():void {  
        dispatchEvent( new SpriteEvent( SHOW_COMPLETE ) );  
    }  
}}
```

---

Wie im Codeausschnitt zu sehen, werden *public*-Funktionen definiert, die das Ein- und Ausblenden ermöglichen und dafür die externe Bibliothek 'TweenMax'<sup>41</sup> verwenden. Diese stellt umfangreiche Funktionalität zur Verfügung, um Eigenschaften von Objekten zu animieren.

In diesem Fall wird die Eigenschaft '*alpha*' (Transparenz) von 0 bis 1 oder umgekehrt, innerhalb einer definierte Zeit, berechnet und somit ein Überblendungseffekt erzeugt.

Die Klasse 'Screen' implementiert Funktionen, die Basiselemente erstellen, positionieren und füllen.

---

### **shareit.view.screens.Screen - reduziert**

---

```
public class Screen extends components.display.Sprite {  
    protected var _minWidth:int = 100;  
    protected var _minHeight:int = 100;  
    public var edgeRadius:int = 20;  
    public function drawBackground( width=200, height=100 ):void {  
        width < _minWidth ? width : _minWidth;  
        height < _minHeight ? height : _minHeight;  
        background.graphics.drawRoundRect( ... );  
    }  
}
```

---

41 <http://www.greensock.com/tweenmax/>

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
protected function updatePosition():void {
    x = Math.round(( ShareIt.WIDTH - this.width ) / 2 );
    y = Math.round(( ShareIt.HEIGHT - this.height ) / 2 );
}
public function init():void {
    drawBackground();
    updatePosition();
}}
```

---

Im Konstruktor werden Instanzen von Sprite (Contentcontainer), Label (TextField-Titel) und Button (schließen) erzeugt. Anschließend wird die Funktion 'init()' aufgerufen, dadurch die Elemente mit 'drawBackground()' gezeichnet und im letzten Schritt mit 'updatePosition()' zentriert ausgerichtet.

### 5.6.1.2 Textfeld

Für die Repräsentation von Inhalt (Text) ist die 'Label'-Klasse sehr entscheidend. Sie ermöglicht die Anzeige von Text, indem sie von 'flash.text.TextField' abgeleitet ist und diverse Einstellungen kapselt.

---

#### **Shareit.view.components.text.Label - reduziert**

---

```
public class Label extends TextField {
    public function Label() {
        selectable = false;
        height = 16;
        embedFonts = true;
        defaultTextFormat = new TextFormat( FONT, 12, 0xFFFFFFFF );
    }
}
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
public function set textAlign( value:String ):void {
    // update 'defaultTextFormat'
}
public function set bold( value:Boolean ):void {
    if( value )
        // bold Einstellungen
    else
        // normal Einstellungen
}
public function set size( value:Number ):void {
    // update 'defaultTextFormat'
}}

```

---

Im Konstruktor werden Grundeinstellungen wie Höhe und Textformat vorgenommen, um nach einer Instanziierung von 'Label' und einer anschließenden Textzuweisung, ohne weitere Einstellungen, eine Ausgabe zu erhalten. Die *setter*-Funktionen 'textAlign()', 'size()' und 'bold()' nehmen entsprechende Werte entgegen, lesen 'defaultTextFormat' aus, aktualisieren den jeweiligen Wert und weisen das aktualisierte TextFormat wieder zu. Dieser Funktionsumfang ist für die Anwendung ausreichend und ermöglicht verschiedene Textrepräsentationen.

### 5.6.1.3 Eingabefeld

Die Anwendung „ShareIt“ benötigt nicht nur ein Element zur Textausgabe, sondern auch zur Texteingabe. Dazu wurde die Klasse 'TextInput' von 'Label' abgeleitet. Sie erweitert 'Label' um *setter/getter*-Funktionen zum setzen bzw. auslesen des Textes und überprüfen auf korrekten Inhalt.

### **shareit.view.components.text.TextInput - reduziert**

---

```
public class TextInput extends Label {
    public function TextInput() {
        super();
        type = TextFieldType.INPUT;
    }
    public function get isValid():Boolean {
        if( required ) return validate();
        return true;
    }
    protected function validate():Boolean {
        if( text == "" ) return false;
        else if( regExp )
            return regExp.test( text );
        else
            return true;
    }
}
```

---

Im Konstruktor wird das Attribut 'type' auf 'TextFieldType.INPUT' gesetzt, wodurch die Basisklasse 'TextField' als Eingabefeld definiert wird. Die Funktion 'isValid()' ruft, wenn die Eingabe erforderlich ist (required = true), die Funktion 'validate()' auf. Diese prüft die Eigenschaft 'text' (*getter*-Funktion) auf Leerstring und im Anschluss, wenn eine Regular-Expression (kurz RegExp), z.B. für E-Mail-Adresse, definiert ist, ob die Texteingabe dieser entspricht.

### 5.6.1.4 Schaltfläche

Eine weitere wichtige Klasse, die eine Schaltfläche repräsentiert, ist 'ContentButton'. Sie wird von 'components.controls.Button' abgeleitet und beherbergt Funktionen die die Visualisierung der Schaltfläche bei Mausaktivität entsprechend anpassen.

---

#### **shareit.view.components.buttons.ContentButton - reduziert**

---

```
public class ContentButton extends Button {
    public function ContentButton() {
        super();
        setStyle("textColor",StyleManager.getColor(0x000000));
        setStyle("textOverColor",StyleManager.getColor(0xFFFFFFFF));
        _label.textColor = _styles[ "textColor" ];
    }
    override public function set enabled( value:Boolean ):void {
        super.enabled = value;
        alpha = value ? 1 : .5;
    }
    override protected function onMouseOver( e:MouseEvent ):void {
        TweenMax.to( _background, showDuration, { tint:
            _styles[ "overColor" ] });
        _label.textColor = _styles[ "textOverColor" ];
    }
}
```

---

Im Konstruktor werden mit der geerbten Funktion 'setStyle()' für zwei Zustände (MouseOver und -Out) Farbwerte für das Titel-Textfeld und die Hintergrundfläche definiert, indem ein Dictionary<sup>42</sup> für ein 'key'-Attribut einen Wert erhält und im Anschluss über 'key' ausgelesen werden kann. Die überschriebene Funktion 'onMouseOver()', von 'Button' geerbt, animiert mittels 'TweenMax' die Eigenschaft 'tint'

<sup>42</sup> [http://livedocs.adobe.com/flash/9.0\\_de/ActionScriptLangRefV3/flash/utils/Dictionary.html](http://livedocs.adobe.com/flash/9.0_de/ActionScriptLangRefV3/flash/utils/Dictionary.html)

des Hintergrunds (`_background`) und ändert die Eigenschaft `'textColor'` des Titel-Textfelds (Label). Eine vergleichbare Funktion ist `'onMouseOut()'` die die Eigenschaften auf die entgegengesetzten Werte setzt. Die Funktion `'fitWidthToLabelWidth()'` ermöglicht eine Anpassung des Hintergrunds an das Titel-Textfeld (Label). Dadurch kann der Titel beliebig gesetzt und im Anschluss die Schaltfläche daran angepasst und ausgerichtet werden.

Die Klasse `'Button'` beherbergt des Weiteren diverse *setter*- und *getter*-Funktionen mit denen verschiedene Zustände (z.B. `selected`) und Eigenschaften gesetzt bzw. abgefragt werden können.

### **5.6.1.5 View & Mediator Zusammenspiel**

Am Beispiel von `'ScreenLogin'` und `'MediatorScreenLogin'` wird nun das Zusammenspiel von View und Mediator näher erläutert.

Die Klasse `'ScreenLogin'` definiert als *public*-Attribute die Eingabefelder und Schaltflächen, dadurch kann `'MediatorScreenLogin'` auf diese direkt zugreifen. Der Konstruktor initialisiert alle Elemente und setzt wichtige Eigenschaften, wie z.B. die RegExp für eine E-Mail. Im Anschluss setzt die Funktion `'init()'` den Hintergrund und positioniert alle Komponenten.

---

**shareit.view.screens.ScreenLogin - reduziert**

---

```
public class ScreenLogin extends Screen {
    public var inputEmail:TextInput;
    public var okBtn:ContentButton;
    public function ScreenLogin() {
        inputEmail.regExp =
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
        /^[A-Za-z0-9][\w.-]+@\w[\w.-]+\.[\w.-]*[A-Za-z][A-Za-z]$/i;
    }
    public function isValid():Boolean {
        var valid:Boolean = true;
        if( !inputEmail.isValid ) { ...
        if( !inputPassword.isValid ) { ...
        return valid;
    }
    public function getLoginData():XML {
        var str:String = "<userdata>" + ...
        return new XML( str );
    }
}
```

---

Die Funktion 'isValid()' wird vom Mediator aufgerufen, prüft nacheinander die relevanten Eingabefelder und setzt das Attribut 'textColor' der vorangestellten 'Labels', im Anschluss wird die Variable 'valid' (Boolean) zurückgegeben. Sind alle Eingaben vollständig und korrekt wird die Funktion 'getLoginData()' aufgerufen, generiert aus den Eingabewerten einen XML-String und gibt diesen als XML zurück.

Der 'MediatorScreenLogin' bildet die Schnittstelle zur View 'ScreenLogin', definiert deren Verhalten und reagiert auf registrierte Notifications.

---

### **shareit.view.mediator.MediatorScreenLogin - reduziert**

---

```
public class MediatorScreenLogin extends MediatorScreen {
    public function MediatorScreenLogin(mediatorName,viewComponent){
        super( mediatorName, viewComponent );
    }
    public function get screenLogin():ScreenLogin {
        return viewComponent as ScreenLogin;
    }
}
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
override public function listNotificationInterests():Array {
    return [ Notify.LOGIN_SUCCESS, Notify.LOGIN_FAULT,
            Notify.LOGIN_ERROR ];
}
override public function handleNotification(notification):void{
    switch( notification.getName() ) {
        case Notify.LOGIN_SUCCESS:
            sendNotification( ...
            break; ...
    }
}
override public function onRegister():void{ super.onRegister();
    screenLogin.okBtn.label = "Login";
    screenLogin.okBtn.addEventListener( MouseEvent.CLICK,
                                        onClickedButton );
}
protected function onClickedButton( e:MouseEvent ):void {
    if( e.currentTarget == screenLogin.okBtn ) {
        if( screenLogin.isValid() ) {
            ProxyLogin.sendData( screenLogin.getLoginData() );
        }
    } else if( e.currentTarget == screenLogin.registerBtn ) {
        sendNotification(...
    }
}
}}
```

---

Die Funktion gibt mit 'listNotificationInterests()' (Array) an, welche Notifications dieser Mediator zugestellt bekommen möchte und definiert mit 'handleNotification()', welche Aktionen bei Zustellung ausgeführt werden. Die Funktion 'onRegister()' wird aufgerufen, wenn der Mediator von 'ProxyScreenManager' im System registriert wird. Ist dies der Fall werden die Titel der 'Labels' (Textfelder) und Schaltflächen gesetzt und den Schaltflächen Ereignisse (Events) hinzugefügt. Als Callback-Funktion wird

'onClickButton()' registriert. Wird eine Schaltfläche geklickt erfolgt der Aufruf dieser Funktion und eine Überprüfung der Eigenschaft 'currentTarget' des Events (e), welche die geklickte Schaltfläche referenziert. Handelt es sich um den 'okBtn', wird 'screenLogin.isValid()' aufgerufen. 'screenLogin' ist eine getter-Funktion, die die Variable 'viewComponent' (Object), die dem Mediator übergeben wurde, auf 'ScreenLogin' castet. Sind die Eingaben korrekt, wird der Funktion 'sendData()' des 'ProxyLogin' der Rückgabewert von 'screenLogin.getLoginData()' übergeben und vom Proxy werden die Daten an ein Serverskript gesendet. Der 'ProxyLogin' sendet nach dem Empfang einer Serverantwort eine Notification und der 'MediatorScreenLogin' reagiert entsprechend.



Abbildung 1: PureMVC-ScreenLogin

### **5.6.1.6 Erstellte Views & Mediatoren**

Die PureMVC Anwendung „ShareIt“ benötigt verschiedene Views um die unterschiedlichen Zustände zu repräsentieren. Nachfolgend werden die erzeugten View und Mediator-Kombinationen aufgelistet und deren Funktionsweise beschrieben.

Diverse Daten müssen, bevor die Anwendung verwendet werden kann, geladen sein. Zur Anzeige dieses Vorgangs wird der 'ScreenPreloading' mit dem

'MediatorScreenPreloading' eingesetzt. Der Mediator bekommt vom 'ProxyPreloader' mitgeteilt in welchem Stadium (Progress, Success und Fail) sich der Ladevorgang befindet und aktualisiert entsprechend den Screen.

Der 'ScreenRegister' nimmt Nutzereingaben (Name, Adresse und Kontaktdaten) entgegen und prüft diese auf Korrektheit, dabei stellt der 'MediatorScreenRegister' die Schnittstelle zum System dar und leitet die Daten an den 'ProxyRegister' weiter. Die Funktionalität gleicht der von 'ScreenLogin' und 'MediatorScreenLogin'.

Den Contentbereich der Anwendung wird von 'ScreenOverview' und 'MediatorScreenOverview' repräsentiert. Dabei beinhaltet der Screen eine Instanz der Klasse 'SimpleUserProfil', die in Kurzform die Nutzerdaten und eine bearbeiten-Schaltfläche anzeigt, und Schaltflächen zum anlegen eines Tauschobjekts und anzeigen der Tauschobjekte beinhaltet.



Abbildung 2: PureMVC-ScreenOverview

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

Der Nutzer erhält mit dem 'PopupUserData' die Möglichkeit seine Nutzerdaten zu aktualisieren. Dabei stellt 'MediatorPopupUserData' die Schnittstelle zum System dar, übergibt dem 'ProxyEditUserData' die aktualisierten Daten und reagiert auf Notifications den Aktualisierungsvorgang betreffend.

Möchte der Nutzer ein neues Tauschobjekt anlegen, kann er die relevanten Daten im 'PopupShareObject' eingeben und speichern. Die Daten nimmt der 'MediatorPopupShareObject' entgegen, übergibt sie an den 'ProxySaveShareObject' und reagiert auf entsprechende Notifications.



The image shows a dialog box titled 'Gegenstand' with a close button (X) in the top right corner. The dialog contains three input fields: 'Objekttitel' for the title, 'Objektbeschreibung' for the description, and an empty field for the owner ('Eigentümer'). A 'Freischalten' button is located in the top right area. At the bottom, there are two buttons: 'Speichern' (Save) on the left and 'Abbrechen' (Cancel) on the right.

Abbildung 3: PureMVC-PopupShareObject

## 5.6.2 SproutCore Views

Als Basis aller Views, die von SC standardmäßig zur Verfügung gestellt werden, dient *SC.View*. Sie stellt alle Eigenschaften bereit, die nötig sind, verschiedenste Formulare und Ansichten zu entwickeln. Ihr lassen sich mit der Eigenschaft '*childViews*' (Array) beliebige andere Views hinzufügen.

In der Anwendung „ShareIt“ finden *SC.View*, *SC.ToolbarView*, *SC.LabelView*, *SC.TextFieldView*, *SC.ButtonView*, *SC.SceneView*, *SC.ScrollView* und *SC.ListView* Anwendung.

Dieser Abschnitt stellt aufeinander aufbauend wichtige Views der Anwendung „ShareIt“ vor. Dabei wird auf die Eigenschaften und Verbindungen mit Controllern eingegangen und verschiedene Vorgehensweisen/Realisierungsmöglichkeiten erläutert.

### 5.6.2.1 Hauptoberfläche

SC stellt als Haupt-Interaktionsschnittstelle die Klasse *SC.Page* zur Verfügung. Als zweite Interaktionsebene kommt *SC.Pane* bzw. deren Sub-Klassen (*SC.MainPane*, *SC.MenuPane* oder *SC.AlertPane*) zum Einsatz. In der Anwendung „ShareIt“ wird ausschließlich *SC.MainPane* eingesetzt. Diese Klasse stellt die Möglichkeiten bereit, den Contentbereich frei zu gestalten. Die Interaktionsebenen und verschiedenen Ansichten werden in der Datei 'main\_page.js' angelegt. Das Verzeichnis 'resources' und die darin befindliche 'main\_page.js' werden automatisch angelegt, wenn ein Projekt mittels Ruby-Kommandozeile, mit dem Befehl '*sc-init <projektname>*', erzeugt wird. Nachfolgend ist eine vereinfachte 'main\_page.js' dargestellt. Im Anschluss wird diese ausführlich in Einzelschritten erläutert.

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

### **share\_it\resources\main\_page.js - reduziert**

---

```
ShareIt.mainPage = SC.Page.design({
  pane: SC.MainPane.design({
    childViews: 'middleView topView bottomView'.w(),
    topView: SC.ToolbarView.design({
      layout: { height: 75 },
      anchorLocation: SC.ANCHOR_TOP,
      ...
    }),
    bottomView: SC.ToolbarView.design({
      layout: { height: 32 },
      anchorLocation: SC.ANCHOR_BOTTOM,
      ...
    }),
    middleView: SC.View.design({
      layout: { left: 0, top: 75, right: 0, bottom: 32 },
      ...
    })
  })
});
```

---

Im ersten Schritt wird 'ShareIt.mainPage' von *SC.Page* abgeleitete (*design*). Im Anschluss erfolgt die Implementation der 'pane', die von *SC.MainPane* abgeleitet wird. (Um diese MainPane zur Anzeige zu bringen, müsste 'ShareIt.mainPage.pane.append();' ausgeführt werden.)

Über die Eigenschaft '*childViews*' (Array) der 'pane' werden die anzuzeigenden Views angegeben. In diesem Fall handelt es sich um zwei *SC.ToolbarView* (Kopf- und Fußbereich) und eine *SC.View* (Contentbereich). Am Ende der Zeichenkette ist die von SC zur Verfügung gestellte Funktion '.w()' angehängt. Sie durchsucht die Zeichenkette nach Leerzeichen, trennt sie in einzelne Zeichenketten und gibt ein Array zurück.

Anstelle der Zeichenkette mit der SC-Funktion kann auch direkt das Array angegeben werden, dies erfordert aber etwas mehr Tippaufwand.

Die Eigenschaft *'layout'* einer View nimmt ein Objekt mit definierten Positionierungs- und Größen-Angaben<sup>43</sup> entgegen und gibt an, wie die View im Browserfenster positioniert wird. Die Angabe der Seitenabstände *left*, *right*, *top* und *bottom* führt dazu, dass die View, bis auf die definierten Abstände zu den vier Seiten, das Browserfenster ausfüllt. Es ist darauf zu achten, dass nur Wertekombinationen angegeben werden, die sinnvoll sind. Zum Beispiel würde *left*, *width* und *right* nicht funktionieren, weil die Breite gleichzeitig dynamische und fest definiert wäre. Des Weiteren ist zu bedenken, dass *'layout'*-Angaben verschachtelter Views aufeinander aufbauen, dadurch kann die Positionierung der Views, in vereinzelt Fällen, schwierig ausfallen.

Die zweimal verwendete *SC.ToolbarView* grenzt einen Kopf- und Fußbereich, durch einen standardmäßigen Hintergrund, optisch vom Contentbereich (*middleView*) ab. Die absolute Position der beiden Views wird mit der Eigenschaft *'anchorLocation'*, für die SC verschiedene Varianten<sup>44</sup> zur Verfügung stellt, und den Werten *'SC.ANCHOR\_TOP'* für oben und *'SC.ANCHOR\_BOTTOM'* für unten definiert. Das *'layout'*-Objekt wird nur mit *height* ausgestattet, wodurch diese Views die Browser-Fensterbreite ausfüllen.

---

### **share\_it/resources/main\_page.js - Ausschnitt**

---

```
topView: SC.ToolbarView.design({
  layout: { height: 75 },
  anchorLocation: SC.ANCHOR_TOP,
  childViews: 'titleLabel loggedInLabel introButton createSOButton
allSOsButton ownSOsButton involvedSOsButton holdedOtherSOsButton
allPersonsButton editUserDataButton logoutButton'.w(),
  titleLabel: SC.LabelView.design({
```

43 <http://docs.sproutcore.com/#doc=SC.View&method=layout&src=false>

44 <http://docs.sproutcore.com/#doc=SC.ToolbarView&src=false>

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
        layout: { top: 10, height: 24 },
        controlSize: SC.LARGE_CONTROL_SIZE,
        fontWeight: SC.BOLD_WEIGHT,
        textAlign: SC.ALIGN_CENTER,
        valueBinding: 'ShareIt.paneMainController.headline'
    )),
    loggedInLabel: SC.LabelView.design({
        layout: { right: 11, top: 6, width: 200, height: 36 },
        textAlign: SC.ALIGN_RIGHT,
        escapeHTML: false,
        valueBinding: 'ShareIt.paneMainController.loggedInStr'
    )),
    introButton: SC.ButtonView.design({
        layout: { left: 10, top: 48, width: 100, height: 24 },
        title: 'Intro',
        target: 'ShareIt.paneMainController',
        action: 'onIntro'
    )), ...
})
```

---

Die 'topView' enthält neun Buttons vom Typ *SC.ButtonView* und zwei Textfelder vom Typ *SC.LabelView*. Beide Textfelder beziehen den anzuzeigenden Text mittels *valueBinding* von Variablen im 'ShareIt.paneMainController'.

Die *SC.LabelView* 'titleLabel' erhält keine feste Breite und als Textausrichtung 'SC.ALIGN\_CENTER', das führt dazu, dass sich das Textfeld über die ganze Fensterbreite erstreckt und der Text mittig ausgerichtet wird. Des Weiteren wird 'titleLabel' als bold (*SC.BOLD\_WEIGHT*) und großes Control-Element (*SC.LARGE\_CONTROL\_SIZE*) ausgezeichnet. Die Font-Auszeichnung (Formatierung) kann in zwei Varianten erfolgen. Zum einen, wie an dieser Stelle, über die zur Verfügung stehenden Eigenschaften der View oder mittels CSS.

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

Die zweite *SC.LabelView* 'loggedInLabel' wird auf rechtsbündigen Text eingestellt und stellt HTML formatierten Text (Nutzername, Vor- und Nachname) dar. Dafür ist es notwendig die Eigenschaft 'escapeHTML' auf *false* zu setzen, denn HTML-Tags werden standardmäßig unterdrückt.

Die Funktionsweise der *SC.ButtonView* wird am Beispiel des 'introButton' erläutert. Die Buttons erhalten mittels 'layout' eine Position und Größe. Der Titel wird in diesem Fall statisch gesetzt, könnte aber genauso mit *valueBinding* an eine Variable eines Controllers gebunden werden. Die Eigenschaften 'target' und 'action' definieren, was bei einem Klick geschieht. 'target' gibt an welcher Controller und 'action' welche Funktion aufgerufen wird. Die Unterschiede der anderen Buttons liegen (einzig) in der Größe, Positionierung und der Zielfunktion. Die Darstellung der 'topView' ist damit vollständig beschrieben.

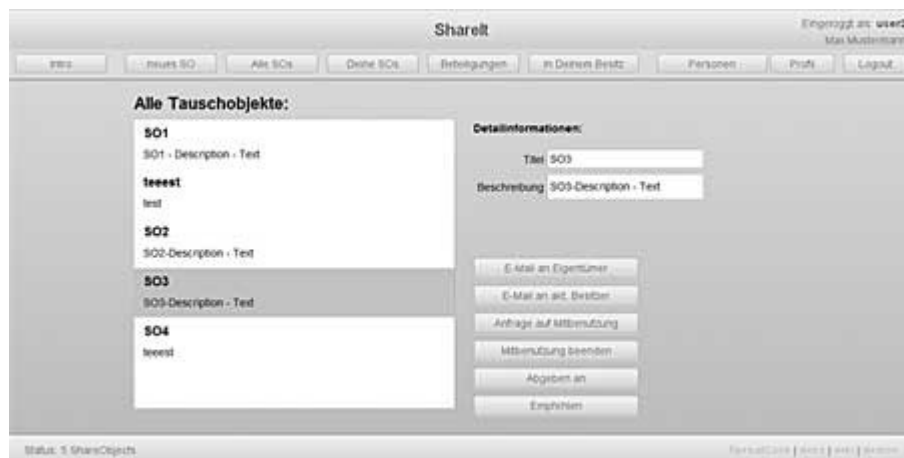


Abbildung 4: SC-ShareIt.mainPage

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

Nun folgt der Code-Ausschnitt der 'bottomView' und anschließend deren Erläuterung. Es werden nur noch Elemente beschrieben, die noch nicht eingehend beschrieben wurden.

---

### share\_it/resources/main\_page.js - Ausschnitt

---

```
bottomView: SC.ToolbarView.design({
  layout: { height: 32 },
  anchorLocation: SC.ANCHOR_BOTTOM,
  childViews: 'statusLabel sproutcoreLink'.w(),
  statusLabel: SC.LabelView.design({
    layout: { centerY: 0, left: 20, right: 20, height: 18 },
    valueBinding: 'ShareIt.paneMainController.status'
  }),
  sproutcoreLink: SC.LabelView.design({
    layout: { centerY: 0, height: 18, right: 20},
    classNames: [ 'a' ],
    textAlign: SC.ALIGN_RIGHT,
    escapeHTML: NO,
    value: "<a href='http://www.sproutcore.com'
          target='_blank'>SproutCore</a> ...",
  })
})
```

---

Die 'bottomView' enthält zwei Textfelder vom Typ *SC.LabelView*.

'statusLabel' zeigt die Anzahl der in einer *SC.ListView* (Tauschobjekte und Personen) befindlichen Elementen an. Aus diesem Grund ist es mit einem *valueBinding*, dass auf die Variable 'status' des 'ShareIt.paneMainController' verweist, ausgestattet. Die Variable 'status' wird vom Controller aktualisiert, wenn sich die 'contentView' der *SC.ScrollView* ändert und die in der angezeigten *SC.ListView* befindlichen Elemente vollständig geladen sind.

'sproutcoreLink' stellt drei externe Links, im Beispielcode ist nur ein Link aufgeführt, dar und definiert diese als statischen Text. Die Formatierung der Links wird dabei über die Eigenschaft 'classNames' (Array) angegeben. SC bündelt alle angelegten .CSS-Dateien und sucht anhand der in dem Array angegebenen Zeichenketten die entsprechenden CSS-Blöcke heraus.

---

**share\_it\resources\styles\link.css**

---

```
a:link { font-weight:bold; color:blue; text-decoration:none; }
a:visited { font-weight:bold; color:silver; text-decoration:none; }
a:focus { font-weight:bold; color:red; text-decoration:underline; }
a:hover { font-weight:bold; color:green; text-decoration:none; }
a:active { font-weight:bold; color:lime; text-decoration:underline; }
```

---

Ein Link wird wie gewohnt mit dem a-Tag definiert. Die Linkzustände werden mit CSS definiert und befinden sich im Verzeichnis 'share\_it\resources\styles' in 'link.css'.

### 5.6.2.2 Contentbereich

Der Contentbereich, die verschiedenen Ansichten der Anwendung, wird von der 'middleView' repräsentiert.

Die 'middleView' wird von der *SC.View* abgeleitet, wodurch beliebig Views hinzugefügt werden können. Positioniert wird die View unter der 'topView' und reicht in der Höhe bis zur 'bottomView'. In der Breite füllt sie die Fensterbreite aus. Die Implementation sieht wie folgt aus:

### **share\_it\resources\main\_page.js - Ausschnitt**

---

```
middleView: SC.View.design({
  layout: { left: 0, top: 75, right: 0, bottom: 32 },
  childViews: 'sceneView'.w(),
  sceneView: SC.SceneView.create({
    scenes: ShareIt.paneMainController.getAllScenes(),
    nowShowingBinding: 'ShareIt.paneMainController.currentScene',
    transitionDuration: 500,
  })
})
```

---

Als einziges *Child* wird hier jedoch nur 'scene' vom Typ *SC.SceneView* hinzugefügt.

Die Klasse *SC.SceneView* animiert einen horizontalen Wechsel zwischen verschiedenen Views. Die Animation wird von SC mit CSS Transitions realisiert. Sollte dies der Browser nicht zur Verfügung stellen, wird die Animation mit JavaScript realisiert.

Die 'SceneView' hält auf der Variabel 'scenes' (Array) die verschiedenen Views, die zur Anzeige kommen können. In diesem Fall ist das Array in den 'ShareIt.paneMainController' verlagert, um im Controller einen besseren Überblick zu gewährleisten. Die Funktion 'getAllScenes()' gibt das Array mit den Pfaden der Views zurück.

Die entsprechenden 'Scene'-Views müssen außerhalb der *SC.MainPane* auf der selben Ebene wie die 'pane' (*SC.MainPane*) definiert werden.

Die Eigenschaft '*isShowingBinding*' verweist auf die Variable 'currentScene' des 'ShareIt.paneMainController' und führt bei Wertveränderung dazu, dass die andere View seitlich in den sichtbaren Bereich bewegt wird.

Die Bewegungsrichtung ist abhängig von der Arrayposition (*SC.SceneView* Attribut 'scenes') der aktuellen und neuen View. Je nachdem, ob die neue View vor oder hinter

der der aktuellen View sitzt, erfolgt die Bewegung von links nach rechts oder umgekehrt. Zusätzlich wird die Eigenschaft *'transitionDuration'* (Dauer der Bewegung) von standardmäßig 200 auf 500 Millisekunden gesetzt.

Damit ist die Hauptoberfläche und der Contentbereich abgehandelt.

Aus Gründen der Übersichtlichkeit wurde der Anmelde-Bereich (Login und Registrieren) in eine extra *'ShareIt.loginPage'* verlagert (*'share\_it/resources/login\_register\_page.js'*). Dadurch wird zum einen eine saubere Trennung vom Contentbereich gewährleistet und zum anderen die einfache Wiederverwendbarkeit des Anmeldebereichs ermöglicht. Der Aufbau der *'ShareIt.loginPage'* ist weitestgehend identisch. Die *'topView'* enthält keine Buttons, die *'bottomView'* keine Statusanzeige und die *'middleView'* hält nur drei Views (*loginView*, *registerView* und *thanksView*) in der *'SC.SceneView'* bereit.

### 5.6.2.3 Formular/Eingabemasken

Bei einem Eingabefeld handelt es sich um ein Textfeld mit Eingabemöglichkeiten.

SC ermöglicht es, dass der Text eines *SC.LabelView* editiert werden kann. Dazu muss die Eigenschaft *'isEditable'* auf *YES* oder *true* gesetzt werden. Wird doppelt auf ein solches Textfeld geklickt, öffnet sich ein Inline-Editorfenster und der Text kann verändert werden. Mit *'return'* (Enter-Taste), keine Mehrzeiligkeit vorausgesetzt, oder Klick in einen anderen Bereich wird die Textänderung übernommen. Soll dem Nutzer ein eindeutig zu erkennendes Eingabefeld bereit gestellt werden, stellt SC dafür eine extra Klasse *SC.TextFieldView* zur Verfügung. Die Eigenschaften *'layout'*, *'classNames'* und *'valueBinding'* sind genauso zu verwenden wie bei einer *SC.LabelView*. Die

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

Textänderung wird durch DataBinding direkt in der Controller-Variable angepasst und kann dadurch direkt verarbeitet werden. Der standardmäßige Rahmen kann über die Eigenschaft *'shouldRenderBorder'* und den Wert *false* deaktiviert werden. Wird das Eingabefeld zur Eingabe eines Passworts verwendet, kann mit der Eigenschaft *'isPassword'* und dem Wert *YES* die Textanzeige in Sternchen konvertiert werden. Ist die Eingabe von mehr als ein paar Zeichen erforderlich, kann *'isTextArea'* auf *YES* gesetzt werden und das Textfeld ist für Mehrzeiligkeit, entsprechende Höhe des Textfeldes vorausgesetzt, ausgelegt.



The screenshot displays the 'ShareIt' application interface. At the top right, it shows the user is logged in as 'user2' (Max Mustermann). A navigation bar contains buttons for 'Intro', 'neues SO', 'Alle SOs', 'Profil', 'Logout', and 'E-Mail'. The main content area is titled 'Deine aktuell gespeicherten Daten:' and contains a form with the following fields and values:

Vorname	Max
Nachname	Mustermann
Straße	Musterstrasse
Hausnummer	1
Stadt	Musterstadt
PLZ	01067
Tel.	
E-Mail	max.mustermann@shareit.c
Nutzername	user2

Below the form is a 'Speichern' button. At the bottom of the page, there is a footer with the text 'SproutCore | docs | wiki | demos'.

Abbildung 5: SC-ShareIt.mainPage.updateUserDataView

#### 5.6.2.4 Listen-Ansicht

Zur Anzeige der Tauschobjekte und Personen wird die Klasse *SC.ScrollView* verwendet. Diese View fügt der Ansicht Scrollbalken hinzu, wenn Höhe und Breite des Inhalts, bei Veränderung der Größe des Browserfensters oder hinzufügen eines Elements, den sichtbaren Bereich überschreiten. Mit der Eigenschaft *'hasHorizontalScroller'* bzw. *'hasVerticalScroller'* kann dieses Verhalten deaktiviert werden. Einer *SC.ScrollView* kann nur über diese Eigenschaft *'contentView'*, die einem Container entspricht, eine andere View hinzugefügt werden. In diesem Fall wird eine *SC.ListView*, die eine Liste von definierten Elementen anzeigt, hinzugefügt. Für die korrekte Funktionalität der *SC.ListView* müssen, wie am folgenden Codeausschnitt zu sehen, mehrere Eigenschaften gesetzt werden. Die beiden *DataBindings* werden im Anschluss genauer beschrieben.

---

#### **share\_it\resources\main\_page.js - Ausschnitt**

---

```
scrollView: SC.ScrollView.design({
  layout: { left: 140, top: 30, bottom: 15, width: 350 },
  hasHorizontalScroller: NO,
  contentView: SC.ListView.design({
    layout: { left: 0, top: 15, bottom: 15 },
    rowHeight: 54,
    exampleView: ShareIt.ShareObjectView,
    recordType: ShareIt.ShareObject,
    contentBinding: 'ShareIt.shareobjectsController.arrangedObjects',
    selectionBinding: 'ShareIt.shareobjectsController.selection',
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
        //selectOnMouseDown: YES,  
        //canDeleteContent: YES,  
        //canReorderContent: YES,  
        //isEditable: YES  
    })  
})
```

---

Die Eigenschaft *'rowHeight'* legt die Höhe eines Listenelementes fest. Die Eigenschaft *'exampleView'* erhält einen Verweis auf eine extern erzeugte View *'ShareIt.ShareObjectview'*, die als BasisView der anzuzeigenden Elemente dient.

Die gesetzten *valueBindings* sind folgendermaßen zu verstehen.

Das *contentBinding* auf die Eigenschaft *'arrangedObjects'* des *'shareobjectsController'*, führt dazu, dass die in der *'content'* Variable des *ArrayController* befindlichen Elemente in der Liste angezeigt werden. Die Variable *'content'* entspricht einem Array von Elementen. Eine Änderung des Array-Inhalts verändert direkt den Inhalt der *SC.ListView*.

Das *selectionBinding* führt dazu, dass der Datensatz (Record) des ausgewählten Elements auf die Variable *'selection'* des *ArrayControllers* gesetzt wird. Diese Variable wird vom *'selectedShareObjectController'*, einem *ObjectController*, als *'contentBinding'* gesetzt und führt dazu, dass die rechts neben der *SC.ListView* befindliche *selectedView* mit Daten gefüllt und angezeigt wird. Die auskommentierten Eigenschaften ermöglichen Funktionalität wie zum Beispiel Drag&Drop und Editierbarkeit eines Elements, die in dieser prototypischen Anwendung nur angedeutet werden.

Die Basisview der Listenelemente (*ShareObjectView*) ist nicht wie alle anderen Views mit *SC-Elementen* erstellt.

---

**share\_it\resources\views\share\_object.js - reduziert**

---

```
ShareIt.ShareObjectView = SC.View.extend( SC.ContentDisplay, {
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
layout: { left: 10 },
classNames: [ 'shareobject-view' ],
contentDisplayProperties: 'title description'.w(),
isHovering: NO,
displayProperties: 'isHovering isSelected'.w(),
mouseEntered: function() {
    this.set( 'isHovering', YES );
},
mouseExited: function() {
    this.set( 'isHovering', NO );
},
render: function( context, firstTime ) {
    var title, description = '';
    var content = this.get( 'content' );
    if ( content !== null ) {
        title = content.get( 'title' );
        description = content.get( 'description' );
    }
    if ( this.get( 'isSelected' ) ) {
        context.setClass( 'shareobject-view-selected',
            this.get( 'isSelected' ) );
    } else {
        context.setClass( 'shareobject-view-hover',
            this.get( 'isHovering' ) );
    }
    // Output HTML. Create inner DIV to show our data
    context = context.begin( 'div' )
    .addClass( 'shareobject-view-topfiller' ).push( '&nbsp;' ).end();
    context = context.begin( 'div' )
    .addClass( 'shareobject-view-title' ).push( title ).end();
    context = context.begin( 'div' )
    .addClass( 'shareobject-view-text' ).push( description ).end();
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
        context = context.begin( 'div' )
        .addClass( 'shareobject-view-bottomfiller' ).push( '&nbsp;' ).end();
        sc_super();
    }
});
```

---

Diese View wird von *SC.View* abgeleitet und erhält zusätzliche Funktionalität von *SC.ContentDisplay* (Mixin).

Die Eigenschaft '*contentDisplayProperties*' (Array), von *SC.ContentDisplay* hinzugefügt, gibt die Attribute des '*recordType*' (*Model*) an, die in der View zur Anzeige kommen. '*displayProperties*' (Array), von *SC.View* geerbt, gibt zwei verschiedene Zustände der View an, wodurch die vier Zustände 'MouseOver', 'MouseOut', 'selektiert' und 'nicht selektiert' abgebildet werden. Eine Wertänderung der in den beiden Arrays angegebenen Eigenschaften bewirkt, dass die View neu gerendert wird. Die Eigenschaft '*isHovering*' wird von den überschriebenen Funktionen '*mouseEntered()*' und '*mouseExited()*' verändert, wodurch die Hintergrundfarbe aktualisiert wird. '*isSelected*' wird von der View intern entsprechend gesetzt, wenn mit der Maus auf die View geklickt wird.

Die Function '*render()*' wird überschrieben und ermittelt anhand der Eigenschaft '*content*' die anzuzeigenden Daten (title, description). Danach wird der Parameter '*context*', der die darzustellenden HTML-Elemente beherbergt, mit entsprechenden Anweisungen und CSS Formatierungen gefüllt. Die CSS-Angaben befinden sich in der Datei '*share\_object.css*' im Verzeichnis '*share\_it/resources/styles*' und beinhalten die Formatierungen für '*title*' und '*text*', den darzustellenden Text, und die vier genannten Zustände, die sich durch unterschiedliche Hintergrundfarbe unterscheiden.

### 5.6.3 Schlussfolgerungen

In PureMVC ist der Akteur View unterteilt, dabei stellt ein Mediator eine Art Schnittstelle zu einer View (GUI) dar. Die View enthält einzig Funktionalität sich selbst betreffend. Jegliche View-Ereignisse werden vom Mediator entgegen genommen und in PureMVC-Ereignisse (Notifications) umgewandelt. Commands führen anschließend Logik relevante Aufgaben aus und teilen Mediatoren mit (vorrangig mittels Notification), dass Daten ggf. aktualisiert bzw. eine Aktion ausgeführt wurde. Um auf Aktualisierungen zu reagieren, kann ein Mediator auf Proxies zugreifen und weitere Notifications versenden.

Die Trennung der Funktionalitäten ermöglicht es Views wiederzuverwenden und unabhängig von der Kernfunktionalität zu erweitern. Darüber hinaus ist der Austausch der View-Technologie möglich, z.B. an Stelle von Flash- werden Flex-Komponenten verwendet.

In SC findet keine Trennung der View statt, dadurch sind die notwendigen Komponenten und Controls direkt mit *Bindings* auf Controller-Eigenschaften verbunden. Weiterführende Funktionalität eines Controllers ist für die View nicht relevant. Die Erweiterung bzw. der Austausch von View-Komponenten ist jeder Zeit möglich, dabei müssen nur entsprechende Controller- und Variablennamen aktualisiert werden, was bei der geringen Komplexität relativ einfach ist.

Die Verknüpfung von View und Controller mittels *DataBinding* stellt eine einfache und intuitive Vorgehensweise dar, wodurch die Anwendungsentwicklung überschaubar bleibt und entscheidend beschleunigt werden kann.

Um in PureMVC zu visualisierende Daten zu aktualisieren, ist der Umweg über Notifications und anschließendes Auslesen von Proxies notwendig, dies ist im Vergleich zur Vorgehensweise in SC nicht sehr effizient bzw. stellt die SC-Implementation eine

einfachere Lösung zur Verfügung.

Schlussendlich ist es eine Frage der Komplexität. Die „ShareIt“ SproutCore Anwendung ist gefühlt übersichtlicher als die Flash Anwendung und entsprechend einfacher zu erweitern.

## **5.7 Implementierung der Controller**

Die Anwendung „ShareIt“ verwendet Controller um zwischen Ansichten zu wechseln und Daten in einzelnen Ansichten zu aktualisieren. In diesem Abschnitt werden die implementierten Controller erläutert und Unterschiede herausgestellt.

### **5.7.1 Flash/PureMVC Controller**

In PureMVC wird die Controller-Schicht von Commands, von denen es zwei Arten gibt implementiert. In der prototypischen Anwendung „ShareIt“ kommen nur SimpleSommands zum Einsatz. Jedes Command muss die Funktion '*execute()*', die als Ergebnis einer Notification aufgerufen wird und als Parameter eine Referenz der Notification erhält, überschreiben. Alle erstellten Commands werden vom 'SimpleCommandShareIt' abgeleitet und erhalten dadurch eine *getter*-Funktion '*proxyApplication()*' um auf Basisdaten Zugreifen zu können.

Nachfolgend werden die erstellten Commands mit der enthaltenen Funktionalität erläutert.

Das 'CommandStartup' wird von der konkreten Facade 'ShareItFacada' mit der

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

Notification 'STARTUP' ausgeführt.

---

### **Shareit.controller.simplecommands.CommandStartup - reduziert**

---

```
public class CommandStartup extends SimpleCommand implements ICommand
{
    override public function execute(notification:INotification):void{
        super.execute( notification );
        var applicationProxy:ProxyApplication =
        new ProxyApplication( "ProxyApplication", new DataApplication());
        ( facade as Facade ).registerProxy( applicationProxy );

        sendNotification( Notify.COMMAND_SWITCH_SCREEN,
                        { screen: "ScreenPreloading" });
    }
}
```

---

Dieses Command registriert, die für die Anwendung wichtigen Proxies (ProxyApplication, ProxyPreloader, ProxyLogin usw.). Im Anschluss wird mit der Notification 'COMMAND\_SWITCH\_SCREEN' veranlasst, dass 'ScreenPreloading' angezeigt wird.

Soll ein neuer Screen (View) angezeigt werden, wird die Notification 'COMMAND\_SWITCH\_SCREEN' mit einem 'body'-Parameter gesendet, was ungefähr wie folgt aussieht:

```
sendNotification( COMMAND_SWITCH_SCREEN, { screen:"ScreenName" });
```

Dadurch wird das 'CommandSwitchScreen' ausgeführt:

---

### **Shareit.controller.simplecommands.CommandSwitchScreen - reduziert**

---

```
override public function execute(notification:INotification):void{
    super.execute( notification );
    var switchScreen:Object = notification.getBody();
    var newScreenName:String = switchScreen.screen.toString();
    if( screenManager.getCurrentScreenName() != newScreenName ){
        switch( newScreenName ) {
            case "ScreenPreloading":
                mediatorScreen = getMediatorAndScreen(
                    "MediatorScreenPreloading", "ScreenPreloading" );
                break; ...
        }
        screenManager.showScreen( mediatorScreen );
    }
}

protected function getMediatorAndScreen( med, scr ):MediatorScreen {
    var screenRef:Object = proxyApplication.getDefinition( scr );
    var s:Screen = new screenRef();
    var mediatorRef:Object=proxyApplication.getDefinition( med );
    var mediatorScreen:MediatorScreen=new mediatorRef( mediator, s );
    return mediatorScreen;
}
}
```

---

Die Funktion '*execute()*' wird aufgerufen und übergibt das Attribut 'screen' des '*body*'-Parameters (Object) der Notification an eine switch-Anweisung, diese erzeugt mit mit '*getMediatorAndScreen()*' eine Mediator-Instanz. Die erzeugte Instanz wird anschließend an die Funktion '*showScreen()*' des '*ProxyScreenManager*' übergeben, der den Mediator registriert und die View zur Anzeige bringt.

Das '*CommandShowPopup*' wird vom '*CommandSwitchScreen*' abgeleitet und

implementiert andere switch-Anweisung-Fälle. Ist die Instanz des neuen Mediators erzeugt wird diese an die Funktion 'showPopup()' des 'ProxyScreenManager' übergeben und eingeblendet. Ein Popup kann neben (über) einem Screen platziert werden, wobei ein neuer Screen alle aktuell angezeigten Elemente deaktiviert und ausblendet.

Zum entfernen eines Popups wird dem 'CommandHidePopup' eine Referenz des Mediators auf dem Parameter '*body*' der Notification übergeben. Diese Referenz wird der Funktion 'hidePopup()' des 'ProxyScreenManager' übergeben.

## 5.7.2 SproutCore Controller

In SproutCore gibt es vier verschiedenen Controller. *SC.Controller* ist die Basis der drei Sub-Klassen *SC.ObjectController*, *SC.ArrayController* und *SC.TreeController*. In der Anwendung „ShareIt“ werden alle außer *SC.TreeController*, der zum Beispiel die Anzeige eines Verzeichnisbaumes in einer *SC.ListView* verwalten kann, verwendet.

### 5.7.2.1 Routes

Für die Funktionalität der Anwendung „ShareIt“ ist die Klasse 'ShareIt.viewStateController' von entscheidender Bedeutung. Soll ein Ansichtswechsel erfolgen muss die Browser-URL aktualisiert werden. Die Veränderung der URL wird registriert, die URL-Parameter ausgewertet und die entsprechende View zur Anzeige gebracht. Dazu wird die Klasse *SC.routes* und eine Helferklasse, die die Routen verwaltet, benötigt. Die Helferklasse 'ShareIt.viewStateController' wird von *SC.Object* abgeleitet und besitzt daher keine unmittelbare Controller Funktionalität.

Die Funktion 'initRoute()' wird von den zwei 'pane'-Controllern, bei Klick auf einen Navigationsbutton, aufgerufen um einen Ansichtswechsel auszuführen (URL-Änderung). Dazu wird die entsprechende Route übergeben und mit '*SC.routes.set('location', route)*' die neue URL gesetzt. (*SC.routes* nutzt '#' zum Trennen der Basis-URL von zusätzlichen Parametern.)

Damit auf die Änderung reagiert werden kann, muss eine entsprechende Funktion registriert werden. Dazu stellt *SC.routes* die Funktion 'add()' bereit:

```
SC.routes.add( <route>, ShareIt.viewStateController, 'onChangeUrl' );
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

Die Funktion erwartet als Parameter die abzufangende Route, die Zielklasse und Zielfunktion (zweite Funktion von 'viewStateController').

Folgende Routen (<route>) wurden definiert:

```
':' = 'http://localhost:4020/share_it' (Standardroute)
':area' = 'http://localhost:4020/share_it#main'
':area/_view' = 'http://localhost:4020/share_it#main/intro'
':area/:view/:id' = 'http://localhost:4020/share_it#main/person/1'
```

Wird auf eine 'content'-Ansicht geroutet und das 'ShareItLoginCookie' ist nicht definiert (Nutzer ist nicht eingeloggt), wird auf die 'LoginView' geroutet. Ist das Cookie definiert, wird 'userid' extrahiert, gespeichert und im Anschluss wird der neue Page-Name ermittelt und zur Anzeige gebracht:

---

### **share\_it\routes.js - Ausschnitt**

---

```
var authCookie = SC.Cookie.find( 'ShareItLoginCookie' );
if ( authCookie == null ) {
    this.initRoute( 'login/login' );
    return;
}
else{
    'appUserUserID' = String( authCookie.value ).split( '_' )[ 1 ];
}
var pagePane = ShareIt.getPath( routeParams.area + 'Page.pane' );
pagePane.append();
```

---

Der neue View-Name (routeParams.view) wird an die Funktion 'showView()' des entsprechenden 'pane'-Controllers (paneLoginRegisterController oder paneMainController) übergeben. Diese ermittelt die Position der neuen View im Array und setzt diese auf die Variable 'curSceneIndex'.

Die zwei genannten 'pane'-Controller beherbergen für den Ansichtswchsel die selbe Funktionalität. Anhand des 'ShareIt.paneLoginRegisterController' wird diese nachfolgend dargestellt:

---

**share\_it\controllers\pane\_login\_register\_controller.js - Ausschnitt**

---

```
showView: function( view ){
    var index = shortcuts.indexOf( view );
    this.set( 'headline', 'ShareIt - ' + shortcuts[ index ] );
    this.set( 'curSceneIndex', index );
}
```

---

Die Funktion 'currentScene()' ist abhängig von der Variable 'curSceneIndex'. Aus diesem Grund wird mittels '*.property( 'curSceneIndex' )*' die Variable überwacht und zusätzlich durch die Angabe von '*.cacheable()*' zwischengespeichert.

---

**share\_it\controllers\pane\_login\_register\_controller.js - Ausschnitt**

---

```
currentScene: function(){
    return scenes[ this.get( 'curSceneIndex' ) ];
}.property( 'curSceneIndex' ).cacheable()
```

---

Das hat zur Folge, dass bei Wertänderung von 'curSceneIndex' die Funktion den zwischengespeicherten Wert aktualisiert. Die Deklaration der Funktion mit '*.property()*' hat noch einen anderen entscheidenden Grund. Die Funktion wird dadurch zu einer '*computed Property*' und kann ohne die Klammern aufgerufen werden. Zum Beispiel wäre im Controller der Aufruf wie folgt möglich: *this.get( 'currentScene' );*

Erst durch die Deklaration als '*computed Property*' kann die Funktion 'currentScene()' mittels DataBinding mit dem Attribut '*nowShowingBinding*' der *SC.SceneView*

verbunden werden.

```
nowShowingBinding: 'ShareIt.paneLoginRegisterController.currentScene'
```

Ändert sich die Variable 'curSceneIndex', berechnet 'currentScene()' den Rückgabewert neu, dadurch wird dieser in der View aktualisiert und in Folge dessen die entsprechende neue View zur Anzeige gebracht wird.

Die in diesem Abschnitt vorgestellten Controller-Funktionen lassen sich in allen SC-Controllern anwenden und werden in nachfolgenden Kapitel-Abschnitten nicht mehr näher erläutert.

### 5.7.2.2 ObjectController

Der *SC.ObjectController* bietet die Möglichkeit auf dem Attribut '*content*' ein Datenobjekt abzulegen, wodurch der Controller zu einem Stellvertreter dieses Objekts wird. Der Zugriff auf die Attribute des Objekts kann dadurch über das Attribut '*content*' des Controllers erfolgen. Diese Funktionalität wird vom 'ShareIt.selectedShareObjectController' genutzt, um zu einem ausgewählten Listenelement zusätzliche Informationen und Funktionalität bereit zu stellen. Das *DataBinding* wird folgendermaßen implementiert:

```
contentBinding: SC.Binding.single( 'ShareIt.  
    shareobjectsController.selection' )
```

Das Attribut '*content*' wird mittels '*contentBinding*' mit dem Attribut '*selection*' des 'ShareIt.shareobjectsController' (SC.ArrayController) verbunden. Dadurch wird

gewährleistet, dass sich der Datensatz (Record) des ausgewählten Listenelementes in 'content' befindet. Die *Binding*-Transformation '*SC.Binding.single()*' bewirkt, dass der Inhalt von 'selection', wenn es sich dabei um ein Array handeln sollte, in ein Objekt umgewandelt wird. Diese Transformation ist notwendig, weil der *SC.ObjectController* nur ein Objekt als 'content' aufnehmen kann.

Das *DataBinding* alleine bewirkt noch nicht die Aktualisierung der 'selectionView'. Um das zu realisieren, muss 'content' überwacht werden. Dazu stellt SC Beobachterfunktionen (Observer) bereit.

Für die Überwachung einer oder mehrerer Attribute gibt es zwei Möglichkeiten. Die erste Möglichkeit ist, mit der Funktion '*addObserver( 'key', 'controller', 'function' )*' zu definieren. Als erster Parameter wird die zu überwachende Eigenschaft, als zweites der Ziel-Controller und als drittes die Ziel-Funktion übergeben.

Die zweite Möglichkeit besteht darin, eine Funktion zu definieren und ihr am Ende mittels *chaining* die Eigenschaft '*observes( 'key' )*' anzuhängen. Dadurch wird die Funktion aufgerufen, wenn sich der Wert von 'key' geändert hat. Die Überwachung mehrerer Attribute ist möglich, indem die Attribute durch Komma getrennt angegeben werden. Die zweite Variante stellt die am häufigsten verwendete Implementation dar. Diese wird im folgenden am Beispiel des 'ShareIt.selectedShareObjectController' und der Überwachung des Attributes 'content' erläutert:

Die Variable 'appUserUserID' wird verkürzt dargestellt. Der Variablen-Aufruf erfolgt normaler Weise wie folgt: *ShareIt.personsController.get( 'appUserUserID' )*

---

**share\_it\controllers\selected\_shareobject\_controller.js - Ausschnitt**

---

```
onChangeContent: function() {
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
var soRecord = this.get( 'content' );
if( soRecord == null ){
    this.set( 'isSelected', NO );
    return;
}else{
    this.set( 'isSelected', YES );
}
var isInvolved = NO;
if( soRecord.get( 'persons' ).indexOf( 'appUserUserID' ) != -1 ){
    isInvolved = YES;
}
this.set( 'isAppUserInvolved', isInvolved );
var isMainOwner = NO;
if( soRecord.get( 'mainOwner' ) == 'appUserUserID' ){
    isMainOwner = YES;
}
this.set( 'isMainOwner', isMainOwner );
this.set( 'title', soRecord.get( 'title' ) );
this.set( 'description', soRecord.get( 'description' ) );
}.observes( 'content' )
```

---

Hat sich der Wert von *'content'* geändert wird als erstes geprüft, ob der Wert null ist. Ist das der Fall, wurde die Selektion aufgehoben und die *'selectedView'* muss ausgeblendet werden. Die Controller-Variable *'isSelected'* ist mit dem Attribut *'isVisibleBinding'* der *'selectedView'* verbunden, wodurch die View nur angezeigt wird, wenn ein Listenelement ausgewählt ist.

```
isVisibleBinding: 'ShareIt.selectedShareObjectController.isSelected'
```

Anschließend wird das Attribut *'persons'* (Array) des Tauschobjekt-Record nach der *'userid'* des Nutzers durchsucht.

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
soRecord.get( 'persons' ).indexOf( 'appUserUserID' )
```

Ist die 'userid' enthalten, ist der Nutzer an dem Tauschobjektes beteiligt und bekommt entsprechend Schaltflächen (*SC.ButtonView*) angezeigt. Die Schaltflächen sind mit '*isVisibleBinding*' mit der Variable 'isAppUserInvolved' der View verbunden.

Um die Tauschobjekt-Daten für Nutzer nur lesbar und dem Eigentümer editierbar anzuzeigen, sind die Eingabefelder (*SC.TextFieldView*) mit der Variable 'isMainOwner' per '*isEnabledBinding*' verbunden.

```
isEnabledBinding: 'ShareIt.selectedShareObjectController.isMainOwner'
```

Im letzten Schritt werden die Tauschobjekt-Daten auf ebenfalls mit *DataBinding* (valueBinding) verbundene Variablen (title, description) gesetzt und somit zur Anzeige gebracht.

Aus Gründen der Komplexität werden die Referenz-Funktionen der Options-Schaltflächen (Attribut '*action*' der *SC.ButtonView*) an dieser Stelle nur angedeutet.

### 5.7.2.3 ArrayController

Der *SC.ArrayController* hat die selbe Funktionalität wie der *SC.ObjectController*, er nimmt als '*content*' aber ein Array von Objekten entgegen. Dadurch kann dieser Controller zum befüllen einer *SC.ListView* (abgeleitet von *SC.CollectionView*) verwendet werden. Dafür muss '*content*' der *SC.ListView* mittels '*contentBinding*' mit dem Attribut '*arrangedObjects*' des entsprechenden *SC.ArrayController* verbunden werden. Das Attribut '*arrangedObjects*' veranlasst die *SC.ListView* direkt die im Array befindlichen Objekte anzuzeigen. Das Attribut '*selection*' des Controller wird mit '*selectionBinding*' mit '*selection*' der *SC.ListView* verbunden. Dadurch wird der Record

des selektierten Elements auf dem *'selection'* des *SC.ArrayController* gespeichert. Dieses Attribut kann ein *SC.ObjectController* mittels *Binding* abfragen und zusätzliche Informationen zum ausgewählten Element bereitstellen.

#### **5.7.2.4 Erzeugte Controller**

Die Anwendung „ShareIt“ erfordert verschiedene Controller um die unterschiedlichen Funktionalitäten strukturiert und getrennt abzuarbeiten. Nachfolgend werden die erzeugten Controller aufgelistet und deren Aufgabe und Funktionsweise kurz beschrieben.

Die *SC.SceneView* (Contentbereich) der *'ShareIt.loginPage'* und *ShareIt.mainPage'* verwalten *'ShareIt.paneLoginRegisterController'* und *'ShareIt.paneMainController'*. Beide sind von *SC.Controller* abgeleitet und nehmen Klick-Ereignisse der Schaltflächen entgegen und initialisieren entsprechende Ansichtswechsel (*routes*).

Der *'ShareIt.loginController'* und *'ShareIt.registerController'* sind von *SC.Controller* abgeleitet und verwalten Nutzereingaben. Bei auftretenden Fehlern (fehlende oder falsche Eingabe) wird eine Variable *'errorMessage'* mit entsprechendem Text befüllt und mittels *valueBinding* auf eine *SC.LabelView* zur Anzeige gebracht.

Der *'ShareIt.loginController'* prüft die Nutzereingaben und routet bei Übereinstimmung mit einem *'ShareIt.personStore'*-Record (*Model* *ShareIt.Person*) auf *'main/intro'*. Zur Überprüfung der Login-Daten wird die Funktion *'isUser()'* des *'ShareIt.personsController'* aufgerufen. Befindet sich der Nutzer in der Datenbank, wird mit *SC.Cookie* ein Cookie erzeugt.

---

**share\_it\controllers\login\_controller.js - Ausschnitt**

---

```
writeAuthCookie: function( token ){
    var authCookie = SC.Cookie.create();
    authCookie.name = 'ShareItLoginCookie';
    authCookie.value = token;
    authCookie.expires = null; // Cookie removed when browser closed
    authCookie.write();
}
```

---

Das *SC.Cookie* erhält als *'value'* einen String, der sich aus der Zeichenkette *'cookie'*, einem Unterstrich und der *'userid'* des Nutzers zusammensetzt. Anhand des Unterstrichs kann, nach geschlossener und wieder geöffneter Anwendung, aus dem Cookie die *'userid'* extrahiert werden.

Der *'ShareIt.registerController'* nimmt die Anschrift und Kontaktdaten eines neuen Nutzers entgegen und legt bei Vollständigkeit einen neuen *'ShareIt.Person'*-Record im *'ShareIt.personStore'* an.

---

**share\_it\controllers\register\_controller.js - Ausschnitt**

---

```
createUser: function(){
    var person;
    person = ShareIt.personStore.createRecord( ShareIt.Person, {
        userid: ShareIt.personsController.getNewID(),
        username: this.get( 'username' ),
        forename: this.get( 'forename' ),
        ...
        confirmed: false
    });
}
```

}

---

Der neue 'ShareIt.Person'-Record wird direkt im 'personStore' erzeugt, wodurch eine sofortige Synchronisation mit der Datenbank erfolgt. Das Attribut 'confirmed' wird von der Datenbank-Filterfunktion 'allPersons' abgefragt und verhindert, dass ein nicht freigeschalteter Nutzer die Anwendung benutzen kann bzw. in der Ansicht 'personsView' angezeigt wird.

Der 'ShareIt.createAndEditShareObjectController' ist abgeleitet von *SC.Controller* und nimmt Nutzereingaben (title, description) entgegen. Sind die Daten vollständig, wird ein neuer 'ShareIt.ShareObject'-Record erzeugt und im 'ShareIt.soStore' angelegt. Solange die 'shareobjectView' geöffnet bleibt, können die Daten editiert und nochmals gespeichert werden.

Der 'ShareIt.shareobjectsController' ist abgeleitet von *SC.ArrayController* und zuständig für die Liste der Tauschobjekte in der 'ShareIt.mainPage.shareobjectsView.contentView' (*SC.ListView*). Durch Klick auf eine der Schaltflächen (allSOsButton, ownSOsButton, involvedSOsButton und holdedOtherSOsButton) in der 'ShareIt.mainPage.pane.topView', wird in diesem Controller eine entsprechende Filterfunktion aufgerufen:

- showAllShareObjects()
- showUserShareObjects( userid )
- showUserInvolvedShareObjects( userid )
- showUserHoldedShareObjecty( userid )

Bis auf 'showAllShareObjects()' erhalten alle Funktionen den Parameter 'userid' der auf der Controller-Variable 'userid' gespeichert wird. Anschließend setzt jede Funktion die Variable 'curContentState', die von der Funktion 'updateCurrentContent()' überwacht

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

wird, auf einen Wert aus dem Array 'contentStates'.

---

### **share\_it\controllers\shareobjects\_controller.js - reduziert**

---

```
updateCurrentContent: function(){
    var soQuery = ShareIt.SHAREOBJECT_QUERY;
    var shareobjects = ShareIt.soStore.find( soQuery );
    switch( this.curContentState ){
        case this.contentStates[ 0 ]: // all
            index = 0;
            this.set( 'userid', '' );
            break;
        case this.contentStates[ 1 ]: // user_own
            index = 1;
            var userid = this.get( 'userid' );
            var list = [];
            shareobjects.filterProperty( 'mainOwner',
String( userid ) ).forEach( function( record ) {
                list.push( record );
            }, shareobjects );
            shareobjects = list;
            break;
        case this.contentStates[ 2 ]: // user_involved
            ...
            break;
        case this.contentStates[ 3 ]: // user_holded
            ...
            break;
    }
    this.set( 'content', shareobjects );
}.observes( 'curContentState' )
```

---

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

Ändert sich der Wert von 'curContentState' werden als erstes alle im Store befindlichen Tauschobjekte auf die Variable 'shareobjects' (Array) gespeichert. Anschließend wird eine switch-Anweisung abgearbeitet und 'shareobjects' gefiltert. Am Ende der Funktion erhält das Attribut 'content' das Array als neuen Inhalt.

Ebenfalls am Ende der Funktion wird die Controller-Vairable 'curContentIndex' auf den aktuellen Index gesetzt.

---

### **share\_it\controllers\shareobjects\_controller.js - Ausschnitt**

---

```
currentTitle: function() {  
    return this.get( 'titles' ) [ this.get( 'curContentIndex' ) ];  
}.property( 'curContentIndex' ).cacheable()
```

---

Dadurch wird 'currentTitle' ( computed Property) aktualisiert und entsprechend der Titel über der '*SC.ListView*' aktualisiert.

Für die Anzeige von Zusatzdaten eines ausgewählten Tauschobjekts ist 'ShareIt.selectedShareObjectController' zuständig. Dieser Controller ist abgeleitet von *SC.ObjectController* dessen Implementation im Abschnitt 5.7.2.2 erläutert wird.

Der 'ShareIt.personsController' ist abgeleitet von *SC.ArrayController* und zuständig für die Liste der Personen in der 'ShareIt.mainPage.personsView.contentView' (*SC.ListView*). Dieser Controller beherbergt keine Filter-Funktionen um die *SC.ListView* zu aktualisieren. Stattdessen gibt es Funktionen die Informationen über eine Person bereitstellen. Nachfolgend werden die wichtigsten Funktionen dargestellt und erläutert.

---

### **share\_it\controllers\persons\_controller.js - Ausschnitt**

---

```
isUser: function( username, password ){
```

## 5 Vergleich Flash und HTML5 anhand einer prototypischen Anwendung

---

```
var userid = -1;
inStore = this.get( 'content' ).some( function( person ) {
    if( person.get( 'username' ) == username &&
        person.get( 'password' ) == password ){
        userid = person.get( 'userid' );
        ShareIt.personsController.set( 'appUserUserID', userid );
        return true;
    }
});
return userid;
}
```

---

Diese Funktion wird bei einem Login-Versuch vom `ShareIt.loginController` aufgerufen und vergleicht die Nutzer-Eingaben mit den im `'ShareIt.personStore'` befindlichen Records. Die SC-Filter-Funktion `'some()' (Enumerables)` durchläuft jedes Element (`ShareIt.Person`) in `'content'` und ermöglicht mit einer übergebenen Funktion die Attribute des Record zu überprüfen. Stimmen die Werte überein sind die Login-Daten korrekt, die `'userid'` wird zurückgegeben und der Nutzer wird in den Contentbereich weitergeleitet. Wird kein passender Record gefunden gibt die Funktion `-1` zurück und der `'ShareIt.loginController'` zeigt entsprechend eine Fehlermeldung an.

---

### **share\_it\controllers\persons\_controller.js - Ausschnitt**

---

```
getUserFromUserId:function ( userid ){
    ShareIt.log( 'getUserFromUserId(): userid=' + userid );
    var user = null;
    user=this.get( 'content' ).findProperty( 'userid', String(userid));
    return user;
}
```

---

Die Funktion bekommt 'userid' übergeben und sucht in 'content' nach einer Übereinstimmung mit dem Attribut 'userid' der enthaltenen Records (ShareIt.Person). Dazu wird die SC-Filter-Funktion '*findProperty()*' (*Enumerables*) genutzt. Diese Funktion nimmt als ersten Parameter den Namen des gesuchten Attributs und als zweiten den entsprechend zu findenden Wert entgegen. Zurückgegeben wird bei Übereinstimmung der Record.

Möchte der Nutzer seine Nutzerdaten aktualisieren kommt der 'ShareIt.userDataController' zum Einsatz. Um einen Record verwalten zu können ist dieser Controller von *SC.ObjectController* abgeleitet. Zur Anzeige gebracht werden die Daten, indem 'content' überwacht und bei Wertänderung die mittels *valueBinding* mit 'ShareIt.mainPage.userDataView' verbunden Controller-Variablen mit Werten gefüllt werden. Die Schaltfläche 'speichern' ist nur klickbar (*isEnabledBinding*), wenn der Wert eines der Attribute verändert wurde. Wird auf 'speichern' geklickt, werden die Werte des Record aktualisiert und mit der Datenbank synchronisiert.

### **5.7.3 Schlussfolgerungen**

In PureMVC werden Controller (Commands) erst durch das versenden einer Notification erzeugt und abgearbeitet. Sie bilden die Geschäftslogik der Anwendung ab und sorgen dafür, dass beteiligte Models (Daten) und Views auf den aktuellsten Stand gebracht werden. Ein Command hat dabei keine direkte Verbindung zu einer View (Mediator), kann diese aber über die Facade abfragen bzw. mit einer Notification einen Aktualisierungsvorgang einleiten. Des Weiteren besteht die Möglichkeit mittels MacroCommands auf einander aufbauende SimpleCommands nacheinander

abzuarbeiten.

Im Gegensatz dazu sind SC-Controller direkt mit einer View bzw. Komponenteneigenschaft verbunden. Daraus resultiert, dass sie nicht erst zur Laufzeit erzeugt werden, sondern ab Anwendungsstart präsent sind. Dadurch können übergreifend Funktionen eines anderen Controllers aufgerufen und Anwendungslogik relevante Mechanismen ausgeführt werden. Zusätzlich kann ein Controller als Stellvertreter einer View eingesetzt werden und spezielle Funktionalität bereit stellen. Darüber hinaus bieten Controller die Möglichkeit Variablen mittels prägnanter Anweisungen zu überwachen und auf Wertänderung entsprechend zu reagieren, wodurch mit wenigen Befehlen Automatismen implementiert sind, für die in PureMVC wesentlich mehr Aufwand erforderlich ist.

Beide Konzepte bieten Vorteile, schlussendlich ist es jedoch entscheidend wie der Entwickler mit den Möglichkeiten am besten zurecht kommt. Für große und sehr umfangreiche Projekte, stellt die lose Kopplung der PureMVC-Commands eine strukturierte und modulare Lösung dar. Im konkreten Fall der Anwendung „ShareIt“ ist die dadurch entstandene Komplexität, im Verhältnis zur geringen Funktionalität der Anwendung, sehr hoch und ließ sich mit den SC-Controllern einfacher und strukturiert lösen.

## **5.8 Zusammenfassung und Fazit**

In PureMVC sind die drei Hauptakteure des MVC-Musters noch einmal unterteilt, dies führt zu einer hohen Komplexität der Anwendung. Das SC-Framework verwendet die Akteure strikt ohne zusätzliche Unterteilung, daraus resultiert eine flache Hierarchie auf maximal zwei Ebenen.

In SproutCore ist die notwendige Anwendungslogik überschaubar in eindeutig bezeichneten Controllern platziert und auch nach einiger Zeit ohne viel Einarbeitungszeit verständlich. Dadurch ist es relativ einfach, selbst nach größeren Zeiträumen eine Anwendung zu aktualisieren bzw. zu erweitern. Ein Entwickler der nicht regelmäßig oder gar ständig mit PureMVC arbeitet verliert schnell den Überblick und muss sich mühsam in die Programmstruktur hineinarbeiten, um sinnvolle Anpassungen vornehmen zu können und nicht mehr Schaden als Nutzen anzurichten.

Die Entwicklung der SC „ShareIt“ Anwendung verlief relativ reibungslos und ohne größere Probleme. Aufgetretene Schwierigkeiten ließen sich relativ schnell mit ein paar google-Anfragen und Recherche in der bereitgestellten Dokumentation lösen.

Im Gegensatz dazu gestaltete sich die Entwicklung mit PureMVC und Flash als sehr zeitintensiv und doppelt aufwändig. Es mussten nicht nur die erforderlichen Komponenten und Ansichten entwickelt werden, sondern auch die Verwaltung selbiger und entsprechende übergreifende Logik. Daraus resultiert eine große Zeitspanne, bis die ersten Funktionalitäten überhaupt sichtbar wurden. Mit SC kann mit wenigen Anweisungen ein funktionstüchtiges Ergebnis realisiert werden auf dem Schritt für Schritt aufgebaut werden kann. Dabei wird mit der große Anzahl verschiedener Ansichten (Views) und diverser Funktionalitäten (Build-Tools, DataBinding, DataStore, Routes, Unit-Test, usw. ) eine sehr gute Basis zur Verfügung gestellt um ein Projekt

strukturiert und gekapselt aufzubauen.

Für einen Neueinsteiger stellt das SC-Framework in fast jedem Aspekt die logischere Alternative dar. Es setzt auf den neuen HTML5 Standard, wird stetig weiterentwickelt, bietet ein intuitiv anzuwendendes MVC-Entwurfsmuster, benötigt zu Ausführung kein Browser-PlugIn, die Community bietet für fast alle Probleme eine Lösung und die zur Entwicklung notwendigen Werkzeuge (Framework, Editor und Browser) sind frei verfügbar. Im Gegensatz dazu werden für das PureMVC-Framework keine Funktionalitäten, die über die Kernfunktionalitäten hinaus gehen, bereit gestellt. Einzig ActionScript 3.0 und extern zur Verfügung gestellte Klassen können Entwicklungszeit reduzieren. Die Entwicklung der Anwendung „ShareIt“ mit Flash und dem Architektur-Framework stellte sich als große Herausforderung dar. Nach erfolgter Einarbeitung in das Framework, begann die Entwicklung einzelner Komponenten, wie zum Beispiel skalierbare Ansichtshintergründe, Schaltflächen, Eingabemasken und Listenelemente. Diese Vorarbeiten waren mit erheblichem Aufwand verbunden und reduzierten die geplante Entwicklungszeit der Anwendung enorm. Die verschiedenen Content-Ansichten beschränken sich daher auf rudimentäre Darstellung des Inhalts.

Die Möglichkeit der Verwendung einer gemeinsamen Datenquelle, in beiden Anwendungen, eröffnete sich erst durch die Entwicklung der SC-Anwendung. Erst während der Entwicklung mit SC ist das Datenbankkonzept der CouchDB in die Entwicklung eingeflossen. Durch die Unterstützung der Community konnte dieses Konzept in sehr kurzer Zeit integriert werden und stellte ab dem Moment die Datenbasis der Anwendung dar.

Die Entwicklung des Flash-Prototypen war zu diesem Zeitpunkt so gut wie abgeschlossen. Die Implementation des CouchDB-Datenbankkonzepts würde eine Neuentwicklung der *Model*-Schicht bedeuten und dies war zu dem Zeitpunkt nicht mehr

möglich. Das Ergebnis dieses Vergleichs hätte aber auch diese Veränderung der Flash-Anwendung nicht bewirkt. Das PureMVC-Framework bietet grundlegend zu wenig Funktionalität.

## **6. Ausblick**

Das Internet ist aus der heutigen Gesellschaft nicht mehr wegzudenken. Aus diesem Grund ist es wichtig einen einheitlichen Standard zur Verfügung zu haben, um Webseiten einer möglichst großen Gruppe auf einfachste Weise zugänglich zu machen. Darüber hinaus ist es ebenso wichtig ein reichhaltiges Informationsangebot mit möglichst einer Technologie realisieren zu können, damit die Webseite leicht pflegbar und erweiterbar ist. HTML5 ermöglicht einen auf Semantik optimierten Aufbau einer Webseite, wodurch die Suchmaschinen-Optimierung vereinfacht und eine Webseite eindeutiger hervorgehoben werden kann.

Microsoft hat mit Veröffentlichungen zum Nachfolger von 'Windows 7' die Spekulationen um die Zukunft des Internets gehörig angeheizt. So soll 'Windows 8' die Integration von Apps, mittels HTML5 und JavaScript, erheblich vereinfachen [web 4]. Vielleicht könnte das ein erster Schritt in eine Richtung sein, in der ein multifunktionaler Browser die Arbeit mit einem Computer und dem Internet ermöglicht. Dabei handelt es sich aber nur um eine Spekulation. Die Entwicklungen in diesem Bereich könnten aber maßgeblich dazu beitragen, wie sich das Internet der Zukunft darstellen wird. Die Entwicklungen im Bereich der mobilen Endgeräte (Smartphone) geben einen ersten Vorgeschmack wohin die Entwicklung gehen könnte.

## 6.1 Ist HTML5 ein Flash-Killer?

HTML5 alleine sicher nicht, in Kombination mit einem Framework wie SproutCore ist es aber schon heute möglich, Flash-ähnliches Verhalten zu realisieren. Solange kein einheitlicher Video-Codec festgelegt wird, ist nicht abzusehen, wie sich die Unterstützung von Streaming-Video mit HTML5 entwickeln wird. Flash stellt in diesem Bereich mehrere Jahre Erfahrung, diverse Funktionalitäten bereit, um die Wiedergabe von Videos und Live-Streams zu realisieren. Solange HTML5 dafür keine vergleichbare Antwort liefern kann, wird es Flash in diesem Bereich auch nicht ablösen können. Abgesehen davon bietet HTML5 viele Erweiterungen, die eine Verwendung von Flash, für normale Webseiten, in Frage stellen.

## 6.2 HTML5 und wie weiter?

Alles deutet darauf hin, dass nach der Fertigstellung von HTML5, ein Standard ohne Versionsnummer an dessen Stelle treten wird. Die Webtechnologien entwickeln sich stetig und schnell weiter, aus diesem Grund ist ein HTML6-Version undenkbar, zumal deren Standardisierungsprozess wider mehrere Jahre in Anspruch nehmen würde. Die WHATWG-Richtlinien<sup>45</sup> besagen, dass es kein HTML6 geben wird und nach HTML5 Schluss ist. Demnach wäre es logisch den Bestrebungen nach einer nicht versionierten Entwicklung nachzugehen und einen „lebendigen“ HTML-Standard zu etablieren. Wie sich W3C und WHATWG am Ende einigen werden ist nicht abzusehen und bis es soweit sein wird, werden noch einige Jahre vergehen.

---

<sup>45</sup> <http://blogs.technet.com/b/sieben/archive/2011/03/23/gastartikel-html5-und-dann.aspx>

## **Abkürzungsverzeichnis**

<b>AS</b>	ActionScript
<b>ASCII</b>	American Standard Code for Information Interchange
<b>DOM</b>	Document Object Model
<b>FTP</b>	File Transfer Protocol
<b>MIME-type</b>	Multipurpose Internet Mail Extensions
<b>SC</b>	SproutCore-Framework
<b>Strobe</b>	Strobe Digital Publishing
<b>Telnet</b>	Telecommunication Networks

## Glossar

<b>Compiler</b>	Programm, das aus einer Quelldatei eine lauffähige Binärdatei erstellt
<b>Debugger</b>	Werkzeug, zum Analysieren von Programmcode zur Laufzeit
<b>ECMAScript</b>	standardisierte Fassung von JavaScript
<b>E4X-Spezifikation</b>	Spracherweiterung für ECMAScript
<b>Flex</b>	Entwicklungsframework von Adobe zum Erstellen von RIAs
<b>H.264</b>	Ist ein Standard zur hocheffizienten Videokompression
<b>IntelliSense</b>	automatische Quellcode-Vervollständigung
<b>JavaScript</b>	Skriptsprache für DOM-Scripting in Web-Browsern
<b>JustInTime-Compiler</b>	wandelt Quellcode zur Laufzeit in Maschinen-Code (Bytecode) um
<b>PHP</b>	Skriptsprache für dynamische Webseiten
<b>SGML</b>	Metasprache zum definieren von Auszeichnungssprachen
<b>Smartphone</b>	Mobiltelefon mit Computerfunktionalität
<b>Text-Engine</b>	Bibliothek die diverse Funktionalität rund um Textfelder zur Verfügung stellt
<b>Vektorgrafik</b>	Computergrafik die nicht durch Pixel beschrieben wird, sondern mathematische Funktionen
<b>Videocodec</b>	Ein Codec wird verwendet zum Kodieren und dekodieren von Daten
<b>Webserver</b>	Computer (Dienst) der Dokumente im Internet oder lokalen Netzen bereit stellt

## **Abbildungsverzeichnis**

Abbildung 1: PureMVC-ScreenLogin.....	99
Abbildung 2: PureMVC-ScreenOverview.....	100
Abbildung 3: PureMVC-PopupShareObject.....	101
Abbildung 4: SC-ShareIt.mainPage.....	106
Abbildung 5: SC-ShareIt.mainPage.updateUserDataView.....	111

## **Literaturverzeichnis**

### **[Oreilly 2007]**

Colin Moock: „Essential ActionScript 3.0“

O'Reilly, 2007

### **[adobe 1]**

Adobe Developer Connection, Flash Player Developer Center, SEO Flash

URL: [http://www.adobe.com/devnet/flashplayer/articles/swf\\_searchability.html](http://www.adobe.com/devnet/flashplayer/articles/swf_searchability.html)

[Stand: 25.06.2011]

### **[youtube 1]**

Youtube ApiBlog, News and Notes for Developer, „Flash an the HTML5 <video> Tag“

URL: <http://apiblog.youtube.com/2010/06/flash-and-html5-tag.html>

[Stand: 25.06.2011]

### **[web 1]**

selfhtml.org, Web-Projekts SELFHTML

URL: <http://de.selfhtml.org/intro/internet/www.htm>

[Stand: 25.06.2011]

### **[web 2]**

wi-fom.de, Wissensdatenbank rund um Themen der Wirtschaftsinformatik

URL: [http://winfwiki.wi-fom.de/index.php/HTML5\\_auf\\_mobilen\\_Endger%C3%A4ten](http://winfwiki.wi-fom.de/index.php/HTML5_auf_mobilen_Endger%C3%A4ten)

[Stand: 25.06.2011]

**[web 3]**

webdesigner-magazin.de,

URL: <http://webdesigner-magazin.de/999/news/whatwg-aus-html5-wird-html>

[Stand: 13.05.2011]

**[web 4]**

URL: <http://www.software-dev-blog.de/windows-8-%E2%80%93-html5-statt-silverlight/06/2011/>

[Stand: 25.06.2011]

**[Kroener 2010]**

Peter Kröner: „HTML5 Webseiten innovativ und zukunftssicher“

München, Open Source Press, 2010

**[jolley 1]**

CrunchBase.com, Verzeichnis für Technologiefirmen, Personen und Investoren

URL: <http://www.crunchbase.com/person/charles-jolley> [Stand: 25.06.2011]

**[jolley 2]**

SproutCore-Wiki,

URL: <http://wiki.sproutcore.com/w/page/12413089/What-is-a-Cloud-Application>

[Stand: 25.06.2011]

## **Selbstständigkeitserklärung**

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleich oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Dresden, 28. Juni 2011

.....

Marc Frydetski