Hochschule für
Technik und Wirtschaft
Dresden
University of Applied Sciences

Fachbereich Informatik/Mathematik

Diplomarbeit

im Studiengang Medieninformatik

Thema:

Untersuchung zur Verwendung von Adobe AIR
als plattformübergreifende Laufzeitumgebung
für mobile Applikationen anhand der prototypischen
Umsetzung eines Taskmanagers.

eingereicht von: Silvia Graumann
eingereicht am: 30.09.2011
Betreuer: Prof. Dr. Teresa Merino

Inhaltsverzeichnis

I Einleitung

Zu Beginn der technischen Entwicklung hatten Computer noch die Größe von Kleiderschränken. Im Laufe der Zeit wurden sie immer kleiner und eroberten erfolgreich den mobilen Markt. Sie verschmolzen mit Mobiltelefonen zu einer neuen Kategorie von Geräten, die heute als Smartphones und Tablets bezeichnet werden und nicht mehr aus dem Alltag wegzudenken sind. Das Führen von Telefonaten ist dabei in den Hintergrund getreten. Stattdessen begünstigen Multitouch-Usability und flexible Softwareverwaltung die Nutzung der Geräte als Informations- und Unterhaltungsmedium. Schnellere Internetverbindungen, GPS, HD-Kameras und Beschleunigungssensoren bieten vielfältige Anwendungsmöglichkeiten. Diese werden durch die auf Smartphones und Tablets installierbaren Applikationen – auch Apps genannt – verwirklicht.

■ GPS:
„Global Positioning System" - globales Navigationssatellitensystem zur Bestimmung der Position

■ HD-KAMERAS:
„High Definition" - besonders hochauflösendes Videobild (bis zu 1920 x 1080 Pixeln)

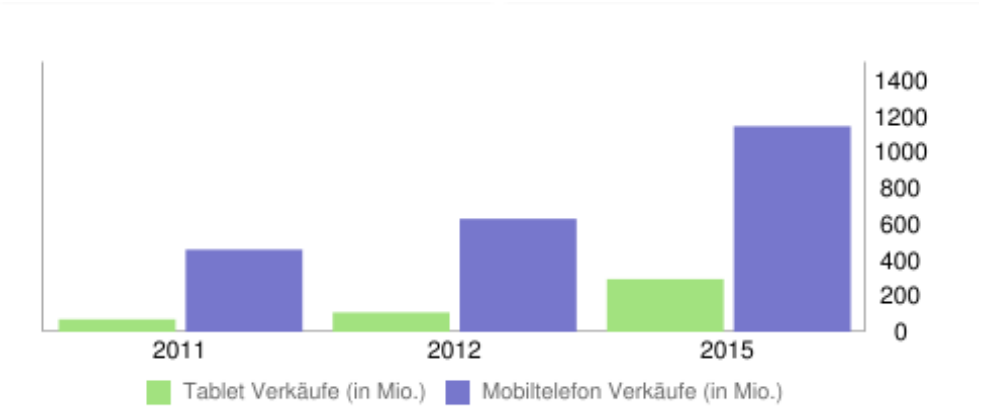


Abbildung 0.1: Wachstum von Tablet- und Mobilphone-Verkäufen pro Jahr bis 2015 in Anlehnung an [Gartner 2011] und [Klingler 2011]

Jene mobilen Applikationen haben in den vergangenen Jahren zunehmend an Bedeutung gewonnen. Dabei ist auch die Zahl der Smartphone-Systeme beträchtlich gestiegen. Eine Vielzahl von Geräteplattformen konkurriert mittlerweile um den größten Marktanteil. Sie unterscheiden sich in ihren Funktionen, Betriebssystemen und Hardware-Komponenten. Insbesondere die Vielfältigkeit der individuellen Programmiersprachen stellt Entwickler und Designer vor neue Herausforderungen.

Waren zunächst vorwiegend iPhone-Applikationen (iOS) gefragt, wird das Interesse an alternativen Systemen wie Android, Blackberry OS oder Win-

dows Phone 7 immer größer. Folglich sind Firmen zunehmend bestrebt, ihre mobilen Anwendungen für mehr als nur eine Geräteplattform anzubieten.

Die starke Fragmentierung mobiler Betriebssysteme bedeutet für die Entwicklung jedoch hohe Aufwände und Kosten. Aufgrund der unterschiedlichen Programmiersprachen und Betriebssysteme muss für jedes System eine individuelle Applikation entwickelt werden.

■ FRAGMENTIERUNG: Zersplitterung, Aufteilung in viele Bruchstücke

Um Entwicklungs- und Wartungskosten möglichst gering zu halten, bietet sich der Einsatz eines Cross-Plattform Frameworks an. Design und Codebasis werden einmal entworfen und für mehrere Zielplattformen kompiliert. Dem User kann so über verschiedene Systeme hinweg ein einheitliches User-Interface bereit gestellt werden.

I.1 Motivation

Eine plattformübergreifende Lösung ist besonders effizient, wenn bei deren Anwendung auf bereits vorhandene Entwicklungskenntnisse zurückgegriffen werden kann. Viele Internetdienstleister entwickeln seit Jahren Webauftritte mit den Webtechnologien HTML, CSS, Javascript und dem browserbasierten Adobe Flash Format. Webtechnologien eignen sich gut als plattformübergreifendes Format, da sie durch die meisten mobilen Internetbrowser gleichermaßen interpretiert werden. Zur Erstellung nativer, plattformeigener Applikationen sind sie jedoch aufgrund mangelnder Systemschnittstellen nicht einsetzbar.

Das Unternehmen Adobe stellt jedoch zusätzlich zum browsergestützten Flash-Plugin ein weiteres Produkt zum Betrachten von Flashinhalten zur Verfügung, die Adobe Integrated Runtime (kurz: AIR).

Adobe AIR ist eines von vielen Cross-Plattform Frameworks. Die Laufzeitumgebung ermöglicht das Ausführen von Flash-Applikationen außerhalb eines Browsers als installierbare Standalone-Anwendungen. So wurde AIR bereits erfolgreich als plattformübergreifendes Format auf Desktopsystemen unter Windows, Linux und MacOS eingesetzt. Im Hinblick auf den mobilen Einsatz wurde der in die Runtime eingebettete Flash Player hin-

■ STANDALONE: Eigenständige Anwendung.

sichtlich Performance und Integration wesentlich optimiert. Die aktuelle Version 10.3 wurde um mobile-spezifische Features wie Multitouch-Gesten erweitert und steht zusammen mit AIR 2.6 bereits ersten mobilen Plattformen zur Verfügung.

Mit dem Einsatz von AIR als Cross-Plattform Lösung auf Mobilgeräten könnten bereits entwickelte Flash-Projekte mit wenigen Anpassungen für Smartphones und Tablets aufbereitet werden. Existierende Online Spiele oder komplexe Businesslösungen müssten nicht für jedes System einzeln nachprogrammiert werden, sondern könnten dem jeweiligen Kunden mit geringem Aufwand zusätzlich für den mobilen Markt angeboten werden.

I.2 Zielsetzung

Der Einsatz eines Cross-Plattform Frameworks richtet sich maßgeblich nach dem Typ der Applikation und ihrer Anforderungen an System- und Userinteraktion. Ob sich dem Entwickler mit Adobe AIR eine ernstzunehmende Alternative zu nativen Einzelentwicklungen bietet, soll Untersuchungsschwerpunkt dieser Arbeit sein.

Dabei wird analysiert, in welchen Fällen die Nutzung von Adobe AIR als plattformübergreifende Lösung für mobile Systeme geeignet ist, wobei sowohl Einschränkungen als auch Stärken der Laufzeitumgebung dargestellt werden. Einen weiteren Fokus bilden die bisherigen Einsatzmöglichkeiten der Runtime bezüglich der aktuell marktführenden Mobilplattformen. Vor- und Nachteile der Verwendung von Adobe AIR sollen anhand eines zu entwickelnden Prototyps untersucht werden. Zudem wird eine Einordnung in den Entscheidungsprozess zum Finden der zweckmäßigsten Lösungsstrategie für die Umsetzung einer mobilen App vorgenommen.

Die Arbeit wird keine Bewertung des Cross-Plattform Ansatzes an sich vornehmen und keinen Vergleich der aufgeführten Cross-Plattform Strategien bezüglich ihrer Vor- und Nachteile führen. Die Untersuchung fokusiert vielmehr die Neugestaltung der Adobe Integrated Runtime hinsichtlich des mobilen Einsatzes.

I.3 Rahmenbedingungen

Auch die Internetagentur „Zeros+Ones – Agentur für neue Medien GmbH“ steigt gegenwärtig in die Softwareentwicklung für Mobilgeräte ein. Immer mehr Kunden beauftragen die Agentur mit einer mobilen Website oder Applikation. Doch nicht nur ihre Kunden, auch die Agentur selbst wird zunehmend mobiler. So soll das aktuelle Zeiterfassungssystem für die Mitarbeiter verbessert werden, wobei eine zusätzliche mobile Applikation die Stundenbuchung von unterwegs oder aus dem Home Office ermöglichen soll. Diese App wird im praktischen Teil der Diplomarbeit prototypisch umgesetzt.

Die von den Angestellten verwendeten mobilen Endgeräte sind in ihren Betriebssystemen ähnlich fragmentiert wie auf dem globalen Markt. Für jedes einzelne System eine eigene Applikation zu entwickeln wäre mit hohen Kosten verbunden. Um den Aufwand für interne Projekte möglichst gering zuhalten, bietet sich für „Zeros+Ones“ der Einsatz eines Cross-Plattform Frameworks an. Aufgrund langjähriger Erfahrung der Agentur im Bereich der Flash- und AIR-Programmierung und der erwähnten Vorteile ist Adobe AIR als Technologie für diese Aufgabe sehr geeignet. Wenn die bereits vorhandenen Kenntnisse auf diesem Gebiet mit wenig Investitionen auf den mobilen Sektor erweitert werden können, würde dies einen großen Vorteil für das Agenturgeschäft bedeuten.

I.4 Aufbau der Arbeit

Die Diplomarbeit ist im Wesentlichen in zwei große Abschnitte gegliedert, wobei sich der erste Teil den theoretischen Grundlagenthemen widmet und der zweite Teil die praktische Umsetzung des Prototyps beschreibt. Im theoretischen Teil wird zunächst das Format der verwendeten Entwicklungstechnologien Adobe Flash und AIR vorgestellt. Dabei wird im Speziellen auf deren geschichtliche Entwicklung im mobilen Sektor eingegangen. Anschließend werden grundlegende Begriffe der mobilen Applikationsentwicklung beschrieben. Dazu zählen die Klassifizierung mobiler Endgeräte sowie die Definition mobiler Apps. Die Analyse aktuell marktführender

mobiler Betriebssysteme im globalen Kontext sowie im Kontext der Agentur „Zeros+Ones“ wird einen weiteren Bestandteil des Abschnitts bilden. Die dabei ermittelten Systeme werden anschließend kurz vorgestellt. Das letzte Theoriekapitel beschäftigt sich mit der Darstellung verschiedener Cross-Plattform Strategien und einer möglichen Einordnung von Adobe AIR.

Der praktische Teil der Arbeit widmet sich zunächst den Anforderungen und Vorgaben des zu entwickelnden Prototyps. Anschließend folgt die Beschreibung der programmiertechnischen Umsetzung der App, wobei genauer auf einzelne Besonderheiten eines mobilen AIR-Projektes eingegangen werden soll. Dies sind zum Einen die modularisierte Projektstruktur der Applikation, zum Anderen das View-Navigator Prinzip, welches der mobilen Screen Methapher zugrunde liegt. Auch die Herausforderung, die Applikation dynamisch an eine Vielzahl verschiedener Displayauflösungen anzupassen, wird in diesem Kapitel eine Rolle spielen. Abschließend behandelt das Kapitel die Portierungsmöglichkeiten der App für die ausgewählten Mobilsysteme. Aus den praktischen Ergebnissen soll im letzten Teil der Arbeit eine Bewertung bezüglich der initialen Zielstellung abgeleitet werden.

II Grundlagen: Adobe AIR im mobilen Kontext

II.1 Adobe® Flash® und AIR®

II.1.1 Adobe Flash

II.1.1.1 Format und Technologie

Adobe® Flash® Player

Hinter dem Autorenwerkzeug „Flash“ verbirgt sich ein umfangreiches Framework zur Erstellung interaktiver, multimedialer Inhalte für das Internet. Vektor- und Bitmap-Animationen können über eine integrierte Zeitleiste erstellt, Audio- und Videoinhalte abgespielt und manipuliert werden. 3D-Szenen und Nutzerinteraktionen mittels Maus, Gesten, Webcam oder Mikrofon sind ebenso Bestandteil des Frameworks wie die dynamische Datenverwaltung mittels XML oder Webservices.

Ursprünglich von FutureWave als einfaches Vektoranimationsprogramm entwickelt, wurde das damalige „FutureSplash“ 1996 von Macromedia übernommen und als Macromedia Flash v1.0 veröffentlicht. Im Jahr 2005 akquirierte Adobe Macromedia und publizierte zwei Jahre später zusammen mit der Adobe Produktreihe CreativeSuite v3 die neunte Version des Flash Players. Durch den Einbau einer neuen Virtual Machine (AVM2) wurde der Flash Player im Zuge der Übernahme auf ein neues Niveau gehoben und für die Integration mit den Adobe Produkten Photoshop und Illustrator aufbereitet.

■ VIRTUAL MACHINE: Modul, welches die Beschreibung einer Berechnung als Input nimmt und diese Berechnung durchführt.

Mit der Verbreitung von Flash im Internet stiegen auch die Anforderungen an dessen Funktionsumfang. Daher bevorzugen Flash-Developer zum Entwickeln aufwendigerer Anwendungen, wie Spiele oder dynamischer Datenvisualisierung, die zugrundeliegende Programmiersprache ActionScript. ActionScript hat sich seit seinen Anfängen als prozedurale Frame-Scriptsprache zu einer vollwertigen objektorientierten Programmiersprache, vergleichbar mit Java und C#, entwickelt. Eine strukturierte API stellt objektorientierte Features wie Vererbung, Interfaces und Event-Dispatching zur Verfügung. Auch klassische Design Patterns, wie das MVC-Prinzip, lassen sich leicht implementieren [Leggett 2006, S.7f].

■ API: „Application Programming Interface“ - Anwendungsprogrammierungsschnittstelle

■ MVC: Das Model-View-Controller Pattern ist ein beliebtes Architekturmuster in der Software-Entwicklung.

Die bereitgestellte Entwicklungsumgebung „Adobe Flash Professional" verarbeitet Flashprojekte im sogenannten FLA-Format. Publiziert wird das in Bytecode kompilierte Projekt anschließend im sogenannten „Small Web Format" (SWF). Dieses wird typischerweise, eingebettet in HTML, über das browsereigene Flash Player Plugin angezeigt. Alternativ können sie in einem Standalone Flash Player abgespielt werden oder zusätzlich als selbstausführbares Projektor-File exportiert werden (.exe unter Windows, .dmg unter MacOS). Laut Adobe erreicht die Verbreitung des Flash Player Plugins auf dem gesättigten Markt fast 99 Prozent aller internetfähigen Desktops [Adobe04 2011].

■ SWF: ursprünglich bekannt durch Director als „Small Web Format", heute als „Shock Wave Flash" bezeichnet

■ GESÄTTIGTER MARKT: Die Statistik bezieht sich auf folgende Länder: United States, Canada, United Kingdom., France, Germany, Japan, Australia, and New Zealand

Als proprietärer Standard konkurriert Flash mit offenen Alternativen wie HTML5 und AJAX. Um diesem Wettbewerbsnachteil entgegenzuwirken, gründete Adobe 2009 das „Open Screen Project", welches von knapp 70 Industriepartnern, zu denen ARM, Nokia, Intel, Motorola und MTV Networks zählen, unterstützt wird. Die Zusammenarbeit soll, unter Nutzung der Vorteile des Adobe Flash Players und zukünftig Adobe AIR, eine konsistente Laufzeitumgebung schaffen, mit deren Hilfe Barrieren beim geräteübergreifenden Veröffentlichen von Applikationen für Desktops, Mobilgeräte, TV-Geräte und andere Verbraucherelektronik abgeschafft werden sollen [Elrom 2009, S. 32].

■ PROPRIETÄR: Herstellergebundene, unveränderbare, unfreie Software. Nicht zwangsläufig kommerziell.

■ AJAX: Asynchrones Übertragungskonzept zwischen Browser und Server.

II.1.1.2 Flash auf mobilen Geräten

Bereits 2003 entwickelte Macromedia eine für mobile Geräte optimierte Variante des Flash Players: Flash Lite. Ursprünglich als „Pocket PC Flash" für PDAs eingeführt, formte sich Flash Lite schnell zu einer vollwertigen Software für mobile Telefone, portable Medienplayer und Spielekonsolen. Der neue Trend des ‚Mobilen Entertainment' etablierte sich zunächst in Japan und verbreitete sich bald in Europa und den USA. Gelegenheitsspiele, Bildschirmschoner, Hintergrundbilder und animierte Klingeltöne wurden mittels Flash Lite für Hersteller wie Symbian, Windows Mobile und BREW umgesetzt. Angefangen mit Animationen und einfachen Applikationsinhalten (Flash Lite 1.0/1.1) bis hin zu umfangreicheren Features wie Video, Sound, Text und komplexen Algorithmen mittels ActionScript 2 (Flash Lite

■ BREW: „Binary Runtime Environment for Wireless" - Ist ein Betriebssystem für Smartphones.

2.0), erhielt der mobile Flash Player innerhalb weniger Jahre einen deutlichen Entwicklungsschub. [Leggett 2006, S. 427,433]

Mit der Veröffentlichung des Flash Players 10.1 Anfang 2010 schaffte Adobe die Zusammenführung von Mobil- und Desktop-basiertem Player und erstmalig die Ausführung von ActionScript 3 auf mobilen Endgeräten. So ist laut Adobe die Version 10.1 das erste konsistente Release des Open Screen Projects, welches unkomprimiertes „Webbrowsing", eindrucksvolle Applikationen und High Definition Video Streaming plattformübergreifend für Desktops, Tablets, Smartphones und Netbooks ermöglicht. Mehr als 250 Millionen Smartphones werden Analysen zufolge bis Ende 2012 vollen Flash Player Support bereitstellen, darunter Plattformen wie Android, Blackberry, Symbian OS, Palm webOS und Windows mobile. [Adobe01 2010]

■ RELEASE: Freigegebene Softwareversion.

Keine Unterstützung für das Flash-Plugin bietet dagegen Apples iOS. In einem offenen Brief vom April 2010 begründete Apple CEO Steve Jobs die Ablehnung von Flash auf iPhone, iPod und iPad mit mangelnder Sicherheit, Stabilität, Performance, sowie fehlender Multitouch-Integration [Jobs 2010]. Sympathisanten des Flash Players dagegen deuten diese Entscheidung als vorrangig geschäftsorientiert und Monopolisierungsmaßnahme des Apple App Stores. Mit dem Update des Players auf die Version 10.2 im Februar 2011 wirkte das Flash-Team bereits vielen dieser Vorwürfe entgegen und implementierte unter anderem ein neues Videoframework mit Hardwarebeschleunigung. Aktuell arbeitet Adobe nun, unter dem Arbeitstitel „Flash Player Incubator" (Version 11), an einem neuen Set von GPU-beschleunigten 3D-APIs („Molehill"), die komplexe 3D-Grafik plattformübergreifend ermöglichen sollen [Adobe03 2011]. Unter diesen Gesichtspunkten bleibt es abzuwarten, ob die Unterstützung unter iOS zukünftig doch ermöglicht werden kann.

II.1.2 Adobe Integrated Runtime – AIR

II.1.2.1 Format und Technologie

Adobe® AIR®

Die Flash-Plattform wird heute oft mit dem Begriff der Rich-Internet-Applications (RIA) verbunden. Damit wird auf multimediale Webinhalte verwiesen, die eine eindrucksvolle User-Experience schaffen und sich innovativer Interaktionsformen bedienen.

Um dieses Erlebnis auch außerhalb des Browsers als installierte Desktopanwendung bereitstellen zu können, veröffentlichte Adobe im Februar 2008 die Adobe Integrated Runtime. Unter Verwendung existierender Webentwicklungskenntnisse lassen sich mit Adobe AIR Flashinhalte für den Desktop publizieren und offline unter Windows, MacOS und Linux ausführen. Kernkomponente einer AIR-Applikation ist der Flash Player, wodurch „AIR über alle Fähigkeiten [...] des Flash Players verfügt", zuzüglich desktop-spezifischer Erweiterungen. Die integrierte Open-Source Engine WebKit sorgt für die Darstellung von HTML, CSS, Javascript und PDF. [Ehrenstein 2009, S.47f]

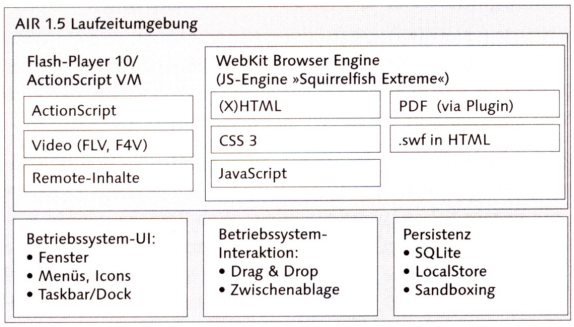


Abbildung 1.1: AIR 1.5 Laufzeitumgebung [Ehrenstein 2009, S.159]

Für die Bereitstellung komplexer Benutzerschnittstellen zur Interaktion mit dem lokalen Dateisystem wurde das ActionScript-Framework um zusätzlich „700 (Komponenten-) Klassen" erweitert und zusammen mit Adobes neuer IDE, dem Flex Builder (heute Flash Builder 4.5) als Flex SDK veröffentlicht [Ehrenstein 2009, S. 61]. Flex verwendet die XML-basierte Sprache MXML, mit deren Hilfe die einzelnen Komponenten beschrieben und angeordnet werden können, wobei die Implementierung der Logik im vertrauten ActionScript erfolgt. Die für „Flash Professional" typische Zeitleiste

■ SDK:
„Software Development Kit" - Sammlung von Werkzeugen, Bibliotheken und Dokumentation einer Software.

■ IDE:
„Integrated Development Environment" - Entwicklungsumgebung

weicht einem neuen zustandsbasierten Modell, womit die Besonderheiten der Desktop Metapher besser abgebildet werden können. Das Flex-Framework und der zugehörige Compiler sind Open-Source, dadurch beschränkt sich die Entwicklung nicht auf ActionScript allein, sondern kann grundsätzlich auch mit den Technologien HTML, CSS, JavaScript und Ajax erfolgen. Flex ist zu einem High-End Tool der Entwicklung robuster Business Applikationen herangereift, insbesondere geeignet für komplexe Datenvisualisierung und Serveranbindungen [Leggett 2006, S.11].

■ DESKTOP METAPHER: Übertragung der Charakteristika einer PC-Benutzeroberfläche, die sich aus Fenstern und Menüs zusammensetzt. Programme basieren auf Zuständen.

Native Funktionen zur Interaktion mit dem lokalem Filesystem und Hardware-Schnittstellen wurden im AIR SDK (aktuelle Version 2.7) zusammengefasst. Bestehende Web-Anwendungen können so mit minimalem Aufwand erweitert und für den Desktop aufbereitet werden. Automatische Update-Funktionen sind ebenso Bestandteil des AIR-SDK, wie unter anderem Drag&Drop Aktionen, Unterstützung der Zwischenablage, Einbindung von lokalem Speicherbereich, Anbindung lokaler Datenbanken (SQLite) und Verwendung von Kontextmenüs. Im Gegensatz zum browserbasierten Flash müssen AIR-Files gepackt, zertifiziert und installiert werden. Zur Ausführung einer AIR-Anwendung wird die AIR Runtime auf dem System benötigt.

■ SQLite: Datenbanksystem, welches sich in einer einzigen Datei befindet.

II.1.2.2 Mobile Erweiterungen des AIR SDK

Am 24. Oktober 2010 stellte Adobe das Update des AIR SDK auf die Version 2.5 vor. Bedeutendste Neuerung war die Unterstützung von Smartphones und Tablets, die auf den Betriebssystemen Android, iOS oder Blackberry Tablet OS basieren.

> „Mit der Laufzeittechnologie Adobe® AIR® 2.5 können Entwickler mit HTML, JavaScript, Adobe Flash® Professional und ActionScript® Web-Anwendungen erstellen, die sich außerhalb eines Browsers als eigenständige Clients ausführen lassen. Mit Adobe AIR, einer Kernkomponente der Adobe Flash-Plattform, verfügen Designer und Entwickler über eine flexible Entwicklungsumgebung, die unzählige Möglichkeiten für die konsistente Bereitstellung von Anwendungen auf mehreren Endgeräten und Betriebssystemen eröffnet. Adobe AIR unterstützt jetzt auch die Betriebssysteme Android™, BlackBerry™ Tablet OS und iOS für mobile Endgeräte [...]“ [Adobe02 2011]

Flash-Inhalte werden auch weiterhin nicht im iPhone-Browser laufen können. AIR Apps für Apple kommen jedoch ohne Flash Player aus und sind daher von diesen Limitierungen nicht betroffen. Auf diesen Aspekt wird im Kapitel II.3 näher eingegangen. Für den Einsatz auf mobilen Endgeräten wurde das AIR SDK um zahlreiche mobile Funktionen erweitert. Im Folgenden wird eine Auswahl von Schnittstellen vorgestellt.

Multitouch-Gesten

Mit Hilfe von Multitouch Events lassen sich sowohl parallel auftretende Finger Taps erfassen als auch gleichzeitige Bewegungen mehrerer Finger auf dem Display. Die Unterstützung von Multitouch Events hängt dabei maßgeblich von der jeweiligen Zielplattform und den unterstützten Hardware-Funktionen ab. Über folgende Eigenschaften lässt sich feststellen, welche Events auf dem Endgerät unterstützt werden:

■ TAP: Einzelner Touch-Point mit dem Zeigefinger.

```
Multitouch.supportsGestureEvents
Multitouch.supportsTouchEvents
```

Listing 1.1: flash.ui.Multitouch.as, Ausgewählte Eigenschaften

Für das Empfangen der Events ist es außerdem notwendig, die Eigenschaft `Multitouch.inputMode` auf einen der folgenden Werte zu setzen:

```
Multitouch.inputMode = MultitouchInputMode.NONE;
Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
Multitouch.inputMode = MultitouchInputMode.GESTURE;
```

Listing 1.2: flash.ui.Multitouch.as, Eigenschaft Multitouch.inputMode

Dabei werden alle Events unter dem Modus `NONE` als Mouse Events interpretiert, unter dem Modus `TOUCH_POINT` ausschließlich als Touch Events und unter dem Modus `GESTURE` als Multitouch-Geste gedeutet. Grundsätzlich unterscheidet die ActionScript 3 API zwischen zwei Multitouch Event Varianten: Touch Events und Gesture Events. Beide werden in der übergeordneten Klasse `Multitouch` gekapselt. Die `TouchEvent` Klasse stellt unter anderem folgende Ereignisse zur Verfügung:

```
TOUCH_BEGIN: Indicates that a touch event has begun.
TOUCH_END: Indicates that a touch event has ended.
TOUCH_MOVE: Indicates that a touch point is being moved
TOUCH_TAP: Indicates that a tap
[Cantrell 2009]
```

Listing 1.3: flash.events.TouchEvent.as, Ausgewählte Konstanten

Die Klasse `GestureEvent` stellt zusammen mit ihren Unterklassen `PressAndTapGestureEvent` und `TransformGestureEvent` folgende Konstanten bereit:

```
GestureEvent.GESTURE_TWO_FINGER_TAP:
//Indicates a gesture defined by tapping with two fingers.
PressAndTapGestureEvent.GESTURE_PRESS_AND_TAP:
//Indicates a gesture defined by a user touching the
screen with one finger, then tapping with another. [...]
TransformGestureEvent.GESTURE_PAN:
//Indicates a gesture to pan content that may be too big
to fit on a small screen.
TransformGestureEvent.GESTURE_ROTATE:
//Indicates a gesture defined by two touch points
rotating around each other in order to rotate content.
TransformGestureEvent.GESTURE_SWIPE:
//Indicates a gesture defined by the quick movement of a
touch point in order to scroll a list, delete an item
from a list, etc.
TransformGestureEvent.GESTURE_ZOOM:
//Indicates a gesture defined by two touch points moving
either toward or away from each other to zoom content in
or out.
[Cantrell 2009]
```

Listing 1.4: flash.events.GestureEvent.as, flash.events.TransformGestureEvent.as

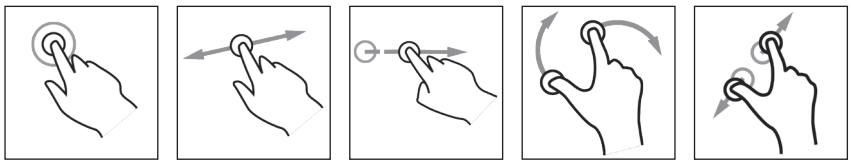


Abbildung 1.2: Visualisierung von Multitouch Events, von links nach rechts: Touch_TAP, GESTURE_PAN, GESTURE_SWIPE, GESTURE_ROTATE, GESTURE_ZOOM [GestureWorks 2011]

Öffnen und Schließen der App auf dem Gerät

Durch das Betriebssystem oder Nutzeraktionen kann eine laufende Anwendung jederzeit deaktiviert werden. In einem solchen Fall sollte der aktuelle Status gespeichert und unnötige Prozesse beendet werden. Wird die Anwendung zu einem späteren Zeitpunkt erneut aktiviert, kann der gespeicherte Status geladen werden. Die AIR API stellt dafür folgende Möglichkeiten zur Verfügung:

```
NativeApplication.nativeApplication.
addEventListener(Event.ACTIVATE, onActivate);
NativeApplication.nativeApplication.
addEventListener(Event.DEACTIVATE, onDeactivate);
NativeApplication.exit()
```

Listing 1.5: NativeApplication.nativeApplication

Beschleunigungssensor

Für viele mobile Smartphoneanwendungen ist die Interaktion mittels Accelerator zu einer gebräuchlichen Eingabemethode geworden. Die Verwendung in ActionScript erfolgt über die Klasse `Accelerometer`:

■ ACCELERATOR: Beschleunigungssensor. Kann die Neigung eines mobilen Gerätes in 3 Achsen bestimmen.

```
var accelerometer:Accelerometer = new Accelerometer();
var isSupported:Boolean = Accelerometer.isSupported;
accelerometer.addEventListener
(AccelerometerEvent.UPDATE, onMove);
accelerometer.setRequestedUpdateInterval(400);
```

Listing 1.6: flash.sensors.Accelerometer.as

Screen Orientation

Eine weitere Möglichkeit, Accelerator Daten auszuwerten, bietet die Screen Orientation API. Diese stellt Methoden zur Erkennung der physikalischen Ausrichtung des Displays, sowie zum Rotieren der `Stage` bereit.

```
stage.setOrientation(orientation:String)
stage.addEventListener
(StageOrientationEvent.ORIENTATION_CHANGE, onChanged);
function onOrientationChanged(e:Event):void{
  trace(e.beforeOrientation, e.afterOrientation)
}
```

Listing 1.7: flash.events.StageOrientationEvent.as, Beispiel

Mit Hilfe der Flash Builder IDE lassen sich zusätzlich zu mobilen ActionScript-Projekten auch mobile Flex-Projekte verwalten. Adobe stellt über diese Option bereits für den mobilen Einsatz optimierte Komponenten zur Verfügung. Mobile Flex-Projekte bieten weiterhin die Möglichkeit, auf die für Mobilgeräte typische Screen Metapher zurückzugreifen. Dafür stellt Flex ein Rahmenprojekt bereit, worüber eine Aufteilung des Screens in sogenannte Views erfolgt, welche wiederum über das Push/Pop-Prinzip in den Fokus navigiert werden können. Auf dieses Thema wird im praktischen Teil der Arbeit noch genauer Bezug genommen.

■ SCREEN METAPHER:
Ein View-basiertes Navigationsmodell, welches durch die User-Interaktion zwischen einer Reihe von Vollbild-Views charakterisiert ist.

Mit dem Release des AIR SDK 2.6 im März 2011 wurden weitere Funktionen ergänzt wie On-device Debugging über USB für Android, Bitmap Capturing für Screenshots und zahlreiche Verbesserungen in der Anbindung von iOS-spezifischen Schnittstellen. Die Version 2.7 brachte hauptsächlich Performance-Verbesserungen für iOS sowie den Export von mobilen Flex-Projekten für iPhone und iPad, für welche bisher lediglich mobile ActionScript Projekte erstellt werden konnten.

■ ON-DEVICE DEBUGGING:
Fehlersuche im Code durch direkte Ausführung der Software auf dem Endgerät.

II.2 Mobiler Kontext

II.2.1 Klassifizierung mobiler Endgeräte

Die Anforderungen an mobile Geräte haben sich in den vergangenen Jahren signifikant geändert. Stand in den 1990er Jahren noch die Telefonfunktionalität und damit die ständige Erreichbarkeit an erster Stelle, so ist heute ein durchgehender und verlässlicher Zugriff auf das Internet mindestens genauso wichtig. Nach der Einführung des ersten portablen Telefons 1973 lassen sich drei relevante Endgerätekategorien unterscheiden:

- Mobiltelefone unterschiedlicher Entwicklungsstufen (1973-2007)
- Touch Phones (> 2007)
- Touch Tablets (> 2010)

Das erste schnurlose Telefon auf dem Markt erinnerte in seiner Größe eher an einen Ziegelstein. Diese Phase wird darum rückblickend auch als „Brick Era" bezeichnet [Fling 2009, S. 3ff]. Die ersten Handys, zu denen auch das „Motorola DynaTAC 8000X" zählt, waren für die Mehrheit der Verbraucher nicht nur unpraktikabel, sondern auch unerschwinglich. Mit der technischen Weiterentwicklung der Mobilfunknetze seit 1988 etablierten sich neue Geräte-Varianten, die schlanker und transportabler waren. Sie vertreten die sogenannte „Candy Bar Era" (1988-1998). Zu dieser Zeit wurde auch der Kurznachrichtendienst SMS eingeführt. Als erster Dienst, der über die bloße Telefonie hinaus ging, fand er schnelle Akzeptanz in der Öffentlichkeit. Mit der Einführung der ersten Kamera-Handys folgte 1998 die Phase der „Feature Phones". Vertreten unter anderem durch das „Motorola RAZR", verfügten sie über größeren Speicher und konnten somit zum Musikhören und Aufnehmen von Digitalfotos verwendet werden. Darüber hinaus ergänzten kleine Applikationen und Services die neuen Features. Die wichtigste Innovation war jedoch die Einführung eines mobilen Browsers zur Nutzung des Internets. Aufgrund hoher Verbindungskosten und geringer Verfügbarkeit mobiler Webseiten wurde dem mobilen Netz zu diesem Zeitpunkt noch keine große Bedeutung beigemessen. Dies änderte sich mit dem Aufkommen der ersten Smartphones im Jahr 2002. Größere Screens, WLAN bzw. Highspeed-Internet und eine komfortable QWERTY Tastatur

■ SMS:
„Short Message Service"

■ WLAN:
„Wireless Local Area Network" - Drahtlose Internetverbindung

bildeten die Grundlage für den mobilen Email-Versand und die aktive Nutzung des Browsers. Auch in der Bedienung gab es erste Neuerungen. So verfügte die Klasse der PDA bereits über ein berührungsempfindliches Display und ließ sich mit einem speziellen Stift bedienen. Trotz dieser Fortschritte erlangten Smartphones zunächst nur im Business-Sektor Beachtung und machten nicht mehr als 15% des globalen Marktanteils aus [Fling 2009, S.9].

■ QWERTY: Amerikanisches Tastatur-Layout. Jede Taste ist nur mit einem Buchstaben belegt.

■ PDA: „Personal Digital Assistent“

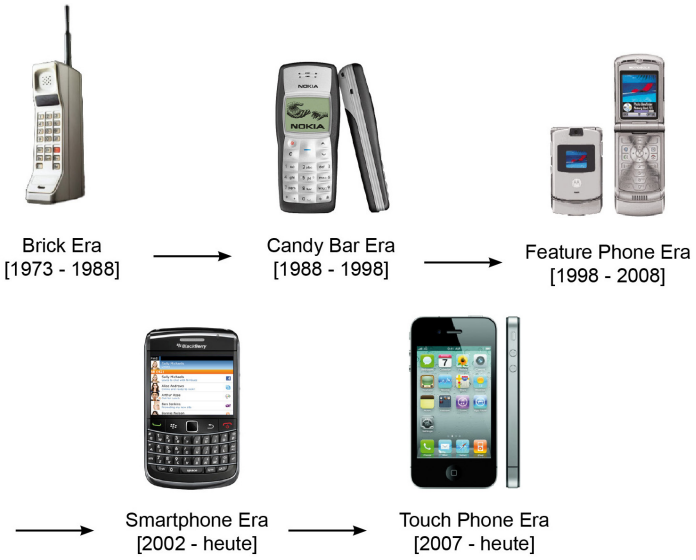


Abbildung 2.1: Entwicklung der Mobiltelefone von 1973 bis heute

Mit der Präsentation des iPhones am 09.01.2007 leitete der Geschäftsführer von Apple Inc., Steve Jobs, die bis heute andauernde Phase der „Touch Era“ ein. Touch Phones werden dabei weder als Telefon noch als Mini-Computer verstanden, sondern repräsentieren ein gänzlich neues Kommunikationsmedium [Fling 2009, S. 12]. Sie stellen sowohl für User-Device-Interaktionen als auch für die Vermittlung von Informationen eine neue, innovative Erlebnisbasis bereit. Ausgestattet mit einem kapazitivem Touch-Display sind die meisten Geräte über Multitouch-Gesten bedienbar. Rotieren von Bildern, Scrollen von großen Inhalten oder das Zoomen in Straßenkarten lässt sich intuitiv über Zweifinger-Wischgesten durchführen. Texteingaben sind per Touch über ein alphanumerisches onscreen keyboard oder über Spracheingabe möglich. GPS-Lokalisierung und ein integrierter Beschleunigungsensor können zur Ermittlung von Standort und der Bewegung des Nutzers verwendet werden. So kann beispielsweise der Aufenthaltsort von

■ ONSCREEN KEYBOARD: virtuelles Multitouch-Keyboard, welches eine herkömmliche Hardware Tastatur ersetzt

Freunden bestimmt werden oder der Weg zum nächstgelegenen Bankautomaten. Touch Phones werden mit angepassten Standard-Betriebssystemen betrieben, welche dem Anwender zahlreiche Applikationen bereitstellen. Die Möglichkeit, selbstständig Apps über einen angebundenen App-Market zu beziehen, ist wohl der größte Unterschied zur Ära der Feature Phones. Mit der Einführung des iPhones fand auch der gewöhnliche Verbraucher Zugang zu den neuen mobilen Endgeräten. Der Begriff „Touch Phone" hat sich jedoch umgangssprachlich nicht durchgesetzt. Stattdessen wird generell der Begriff „Smartphone" gebraucht und darum auch im Folgenden äquivalent verwendet.

■ MARKET:
Allgemeine Bezeichnung eines Vertriebsmarktes für mobile Applikationen.

Am 03.04.2010 wurde in den USA das erste Multitouch-Tablet vorgestellt: das iPad. War die Grundfunktion aller mobilen Geräte bisher die Kommunikation, so sind Touch Tablets mit Display-Größen zwischen A5 und A4 vorrangig auf den Konsum von Multimedia ausgelegt. Sie verfügen über keine Telefonfunktionalität mehr und vollziehen damit den Schritt zum Mini-Computer für Wohn- und Schlafraum. Tablets nutzen vornehmlich Touch-Phone-Betriebssysteme oder angepasste Varianten. Häufig mit Front- und Backkamera ausgestattet, eignen sie sich sowohl als Camcorder als auch für internetgestützte Videokonferenzen. Trotz größerer Displays sind Tablets aufgrund ihres schlanken Designs und geringen Gewichtes sehr portabel. Längere Batterielaufzeiten prädestinieren sie zum Surfen im Internet, Lesen von E-Books oder zum Spielen von Multiplayer-Games. Zukünftig werden sie in Kombination mit TV-Geräten verwendet werden, beispielsweise als Fernbedienung, aber auch zur Darstellung von erweiterten Werbeinhalten. So könnten dem Rezipienten beim Verfolgen eines Fußballspiels im TV-Programm parallel Fanartikel auf dem Tablet zum Kauf angeboten werden.

Abbildung 2.2: Apple iPad und Samsung Galaxy Tab im Vergleich [TECHNISM TODAY 2011]

■ E-BOOK:
Buch in digitaler Form, meistens im PDF-Format.

II.2.2 Mobile Applikationen – Apps

Wie bereits einleitend erwähnt steht der Begriff „App“ für Application (dt. Applikation) und hat sich mit der Einführung des iPhones als gängige Akürzung in der Wirtschaftssprache wie auch im öffentlichen Leben durchgesetzt. Im Allgemeinen sind damit plattformeigene Anwendungen für Touch Phones oder Tablets wie zum Beispiel Spiele, Webbrowser, Kamera und Medienplayer gemeint. Dabei verkörpern Apps keine Programme im herkömmlichen Sinne, wie man sie von Personalcomputern kennt, sondern werden vielmehr als Dienste oder Services verstanden. Erfolgreiche Apps erleichtern dem User einen alltäglichen Vorgang oder unterstützen ihn dabei. Beispielsweise hilft die App „Google Navigator“ dem Nutzer, den kürzesten Weg zwischen seinem aktuellen Standort und der nächsten Tankstelle zu finden.

Nach Fling [Fling 2009, S. 81 ff] lassen sich mobile Applikationen unterschiedlichen Kontexten zuordnen: Utility-, Locale-, Informative-, Productivity- und Immersive Full-Screen Context. Kurze, ergebnisorientierte Dienste, die auf wenig User-Interaktion zurückgreifen, befinden sich im sogenannten „Utility Context“. Solche Apps sind z.B. Wetter-Apps, Wörterbücher und Währungsrechner. „Locale Context“-basiert sind Apps, wenn ihr Informationsangebot auf den Standortdaten des Users beruht. Oft wird bei diesen Apps eine Kartenansicht integriert, in der beispielsweise der Aufenthaltsort von Freunden, die sich in unmittelbarer Nähe befinden, visualisiert wird. Rein informative Inhalte wie Zeitungsartikel oder Kataloge, die kaum User-Input erfordern, beziehen sich auf den sogenannten „Informative Context“. Im Gegensatz dazu unterstehen Dienste mit hoher User-Interaktion, deren Ziel es ist, eine umfangreiche Abfrage möglichst effizient zu bewältigen, dem „Productivity Context“. Dies betrifft z.B. das Finden von Fahrplänen oder Online-Check-In für einen Flug. Spiele und andere Entertainment-Anwendungen, die sich meistens

Abbildung 2.3: Apps werden auf den meisten Geräten durch ein quadratisches Icon repräsentiert

in Vollbild-Darstellung befinden, verweisen auf den „Immersive Full-Screen Context". Der User schenkt der Anwendung seine volle Aufmerksamtkeit und ist nicht durch andere Bedienelemente abgelenkt.

Technisch betrachtet können Apps in verschiedenen Formen auftreten, deren Eigenschaften im Kapitel II.3 genauer beleuchtet werden. In der Regel sind mit dem Ausdruck „App" jedoch sogenannte native (auch „echte") Applikationen gemeint.

Native Apps müssen in der systemeigenen Programmiersprache geschrieben und speziell für die Zielplattform kompiliert werden. Sie haben Zugriff auf sämtliche gerätespezifischen Funktionen und Schnittstellen, wie Dateisystem, Kamera, Mikrofon, Sensoren und lokalen Speicher. Im Unterschied zu mobilen Webseiten können native Apps auch ausschließlich offline eingesetzt werden. Üblicherweise werden sie über einen sogenannten App-Market bezogen und installiert.

II.2.3 Quantitative Verteilung mobiler Betriebssysteme

Als Verbindungsschicht zwischen Computer-Hardware und Anwendungs-Software verwaltet ein Betriebssystem alle Systemprozesse, Ressourcen, Ein- und Ausgaben sowie das Dateisystem. Mobile Betriebssysteme übernehmen ähnliche Aufgaben auf Tablets und Smartphones. Sie sind ressourcenoptimiert und auf die Unterstützung mobiler Features wie WLAN und Breitbandanbindung, GPS-Unterstützung, mobile Multimediaformate und berührungsbasierte Eingaben ausgelegt.

Mobile Betriebssysteme sind stark an ihre Hardware-Plattform gebunden. So wurde beispielsweise das Betriebssystem „iOS" speziell für Apples iPhone und iPad entwickelt. Auch Blackberrys System „RIM OS" ist fest an die Hausmarke gebunden. Dem gegenüber stellte Google Ende 2008 gemeinsam mit der „Open Handset Alliance" das offene Mobiltelefon-Betriebssystem „Android" vor, welches speziell für den Einsatz auf Multitouch-Geräten optimiert wurde. Als freie Software kann es auf verschiedenen Plattformen verwendet werden und wird u.a. von den Herstellern HTC, Motorola,

Samsung und LG eingesetzt [typopark 2010]. Auch andere bedeutende IT-Unternehmen, wie Microsoft und Nokia, versuchen seitdem ihre aktuellen Mobilsysteme für die Touch-Interaktion aufzubereiten.

Als die derzeit dominantesten Wettbewerber gelten Symbian (Nokia), iOS (Apple), Android (Google), Research in Motion (Blackberry) und Windows Phone 7 (Microsoft).

Entwickler sehen sich durch den hohen globalen Konkurrenzkampf mit einer starken und weiter zunehmenden System-Fragmentierung konfrontiert. Die marktführenden mobilen Systeme basieren auf unterschiedlichen Frameworks (SDKs), verwenden verschiedene Programmiersprachen und schreiben individuelle Entwicklungsrichtlinien vor. Während sich Gerätefunktionen wie Kamera, Ortung, Kontaktverwaltung und Datenspeicherung auf den Systemen sehr ähneln, unterscheiden sich die spezifischen APIs, um entsprechende Schnittstellen anzusprechen [Allen 2010, S.5]. Für die Entwicklung nativer Applikationen für X Endgeräte bedeutet dies einen X-fachen Entwicklungs- und Kostenaufwand. Um diese Investitionen möglichst gering zu halten, konzentrieren sich Hersteller oft auf die verbreitetsten Endgeräte und priorisieren die Entwicklung entsprechend der aktuellen Marktanteile der einzelnen Systeme.

II.2.3.1 Globale Marktsituation

Im Jahr 2010 wurde der mobile Markt von vier Wettbewerbern dominiert: Symbian, Android, iOS und RIM. Als langjähriger Markführer mobiler Smartphone-Systeme hält Symbian bis heute die Mehrzahl der globalen

■ OS: kurz für „Operating System“ (dt. Betriebssystem)

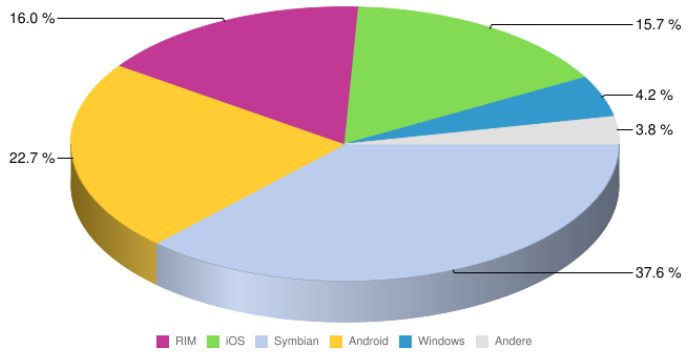


Abbildung 2.4: Globale Marktanteile mobiler Betriebssysteme 2010 [Gartner 2011]

Anteile mit ca. 38%. Seit Beginn der Touchphone-Ära konnte sich Apples iOS mit knapp 16% lange als führendes Multitouch-System durchsetzen, wurde jedoch 2010 von Android mit 23% globalem Marktanteil überholt.

War Symbian Ende 2010 noch die Rolle als Marktführer garantiert, so zeichnen sich mit der Bekanntgabe Nokias, einen Wechsel von Symbian auf „Windows Phone 7" zu vollziehen, bereits pessimistischere Vorhersagen ab. Laut Nokia würde das Symbian OS nicht für die Unterstützung von Touch Phones weiterentwickelt werden, sondern lediglich „bei einfacheren Handys eine Rolle spielen" [taz 2011]. Aktuellen Prognosen des IT-Forschungsinstituts Gartner zu Folge wird Symbians Quote dadurch bis 2015 auf 0% zurückgehen [Gartner 2011]. Stattdessen wird sich durch die Kooperation zwischen Nokia und Microsoft „Windows Phone 7" als neuer Wettbewerber am Markt etablieren und „seinen Marktanteil bis 2015 kontinuierlich auf 19,5% erhöhen" [Hochstätter 2011]. Während iOS weiterhin knapp unter 20% schwankt, wird Blackberrys RIM bis 2015 „hingegen einen Marktanteilsverlust von 16,0 auf 11,0 Prozent hinnehmen" müssen [Hochstätter 2011]. Gartner prognostiziert weiter, dass Androids Marktanteile bereits 2012 auf knapp 50% ansteigen werden. Dies würde durch eine Verschiebung der Zielgruppe begünstigt, wonach zukünftig mehr kostengünstige Mainstream-Geräte mit Android ausgestattet werden sollen.

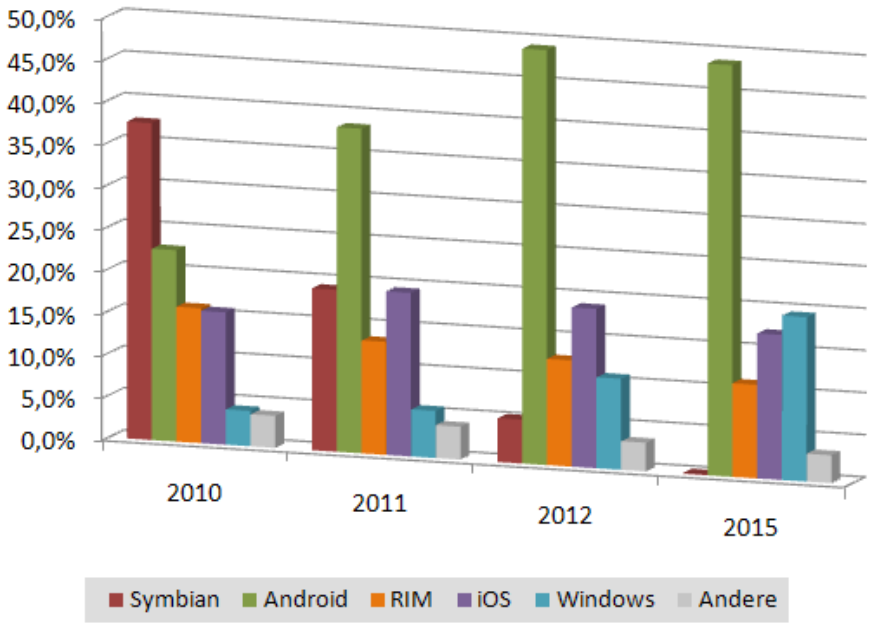


Abbildung 2.5: Gartner-Prognose für Marktanteile bei Smartphone-Betriebssystemen bis 2015 [Hochstätter 2011]

Durch diese Prognosen untermauert, lassen sich zukünftig drei Hauptkonkurrenten auf dem Smartphone-Markt identifizieren: Android mit ca. 48,8%, „Windows Phone 7“ mit ca. 19,5% und iOS mit ca. 17,2%.

Abweichend stellt sich die aktuelle Situation auf dem Tablet-Markt dar. Hier wird iOS mit bis zu 47% globalem Marktanteil wahrscheinlich bis 2015 die beliebteste Tablet-Pattform bleiben. „Keine großen Chancen räumt Gartner MeeGo, Hewlett-Packards WebOS und anderen Systemen ein“ [Klingler 2011]. Google wird mit seinem ersten Tablet-OS „Android 3.0 - Honeycomb“ bis 2015 39% am Tablet-Markt halten. Einzig Blackberrys erstes Tablet „Playbook“ mit dem angepassten Blackberry Tablet OS könnte sich zu einer ernstzunehmenden Konkurrenz zu den beiden Marktführern iOS und Android entwicklen. Gartner sagt Blackberry bis 2015 eine Verkaufssteigerung von 10% voraus.

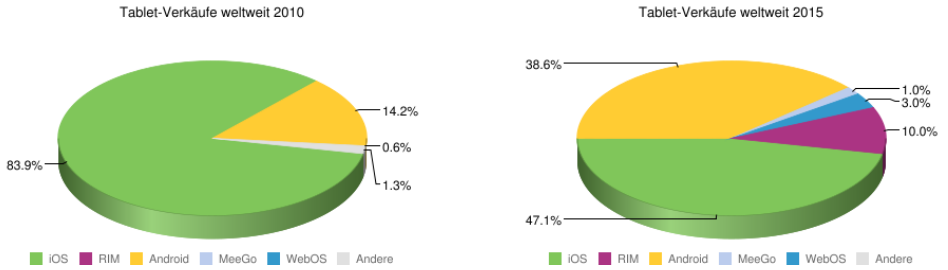


Abbildung 2.6: Marktanteile der Tablet-Betriebssysteme weltweit im Jahr 2010 und 2015. Quelle: Gartner, April 2011 [Klingler 2011]

Schlussfolgernd lassen sich als führende Tablet-Betriebssysteme iOS mit 47%, Android mit 39% und das Blackberry Tablet OS (RIM) mit 10% ausmachen.

II.2.3.2 Verteilung in der Agentur „Zeros+Ones“

Im Rahmen der vorliegenden Diplomarbeit wird, beauftragt durch die Internetagentur „Zeros+Ones - Agentur für neue Medien GmbH“, der Prototyp eines mobilen Taskmanagers entwickelt. Die Taskmanager-App soll später von den Mitarbeitern der Agentur im Arbeitsalltag auf verschiedenen mobilen Endgeräten eingesetzt werden. Die Verteilung der einzelnen Systeme unter den Angestellten ist für die Untersuchungsgrundlage daher ebenso von Bedeutung.

Aus diesem Grund wurde am 17. März 2011 innerhalb der Agentur eine Umfrage zur Verwendung von mobilen Betriebssystemen durchgeführt. Das Ergebnis der Befragung von 54 Teilnehmern (vgl. Abbildung 2.7) zeigt eine hohe Verbreitung von iOS mit 41%, gefolgt von Android mit 20%. Verschwindend gering fällt der Anteil an Symbian-, Windows- und Blackberry-Nutzern aus. Die Dominanz an iPhones lässt sich vermutlich auf den medialen Tätigkeitsbereich der Agentur zurückführen. In diesem Umfeld hielt die Verwendung von Smartphones bereits zu einer Zeit Einzug, zu welcher noch keine bedeutende Alternative zu Apple existierte. Die Betriebssysteme iOS und Android sind somit auch hier die vorherrschenden Konkurrenten. Innerhalb der Agentur lässt sich zum aktuellen Zeitpunkt kein relevantes drittes System erkennen.

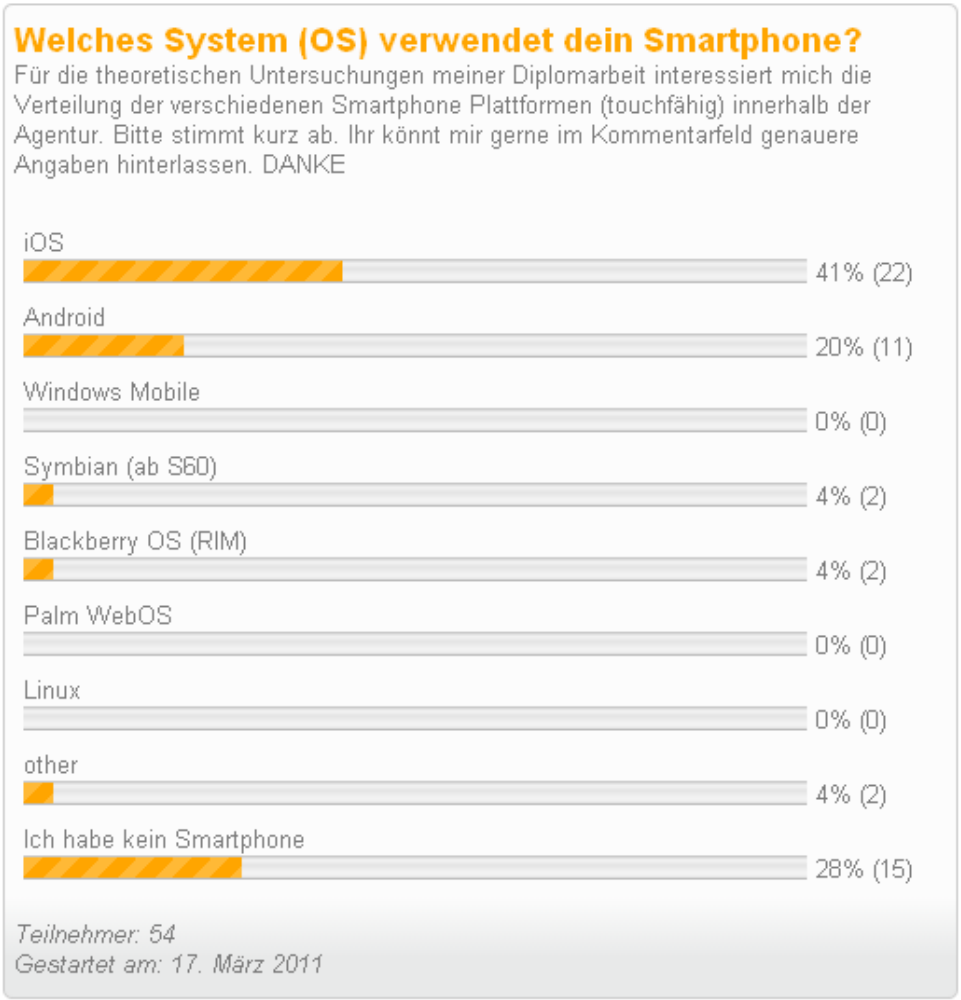


Abbildung 2.7: Agenturinterne Umfrage unter den Mitarbeitern von Zeros+Ones zur Verwendung von mobilen Betriebssystemen am 17.3.2011

II.2.3.3 Auswertung

Fundiert durch die vorangegangenen Statistiken werden für die Diplomarbeit folgende Untersuchungsschwerpunkte aufgestellt:

Für die Portierung des Adobe AIR Prototyps ergeben sich zwei Haupttestsysteme: iOS und Android. iOS aufgrund seiner konstanten globalen Marktanteile und der hohen Verbreitung innerhalb der Agentur, die als Testumfeld für die zu entwickelnde Applikation dient. Android als das langfristig erfolgreichere OS in der Weltwirtschaft. Beide Testsysteme führen sowohl den mobilen Phone- als auch Tabletmarkt an und kommen auf einer Vielzahl von Geräten zum Einsatz. Die relevanten dritten Betriebssysteme unterscheiden sich für Telefone und Tablets. Daher wird „Windows Phone 7“ aufgrund seiner bis 2015 prognostizierten Marktstellung als Smartphone OS herangezogen sowie das Blackberrys Tablet OS für Tablets.

Darüber hinaus soll sich im Verlauf der prototypischen Implementierung der Taskmanager-App zeigen, dass ein positiver Nebeneffekt der Verwendung des Adobe AIR Frameworks darin besteht, die App zusätzlich als Desktop-Widget auf PC und Mac einsetzen zu können. Dies stellt – gerade aufgrund der hohen Anzahl von Nicht-Smartphone-Usern innerhalb des Untersuchungsumfelds – einen großen Vorteil dar.

II.2.4 Anforderungen ausgewählter mobiler Betriebssysteme

Um einen Eindruck von der Entwicklungsweise systemeigener („nativer“) Anwendungen für die einzelnen Betriebssysteme zu vermitteln, sollen diese im Folgenden kurz vorgestellt werden. Dabei wird die Programmiersprache der Systeme näher beleuchtet sowie Richtlinien und Vorgaben erörtert. Der Überblick soll weiterhin den Aufwand gegenüber einer Entwicklung mittels Cross-Plattform-Ansatz veranschaulichen.

II.2.4.1 Betriebssystem und Programmiersprache

Um für iPhone/iPad und iPod Touch zu entwickeln, ist ein Intel-basierter Macintosh mit einer Betriebssystemversion ab OS X v10.5.7 Voraussetzung [Allen 2010, S.17]. Apples IDE XCode bildet die Entwicklungsumgebung und beinhaltet das iOS SDK, einen Device Emulator sowie den sogenannten „Interface Builder“ zum Erstellen der UI-Controls. Das iOS SDK beinhaltet die Klassen des Cocoa Touch Frameworks, eine Erweiterung des Cocoa Frameworks von MacOS X. Die Programmierung erfolgt in der eigens von Apple entwickelten Programmiersprache Objective-C, die einen objektorientierten Ansatz der Ur-Sprache C darstellt. Auch die Integration von C/C++-Code ist möglich.

■ EMULATOR: Software-Nachbildung eines realen Gerätes.

■ UI-CONTROL: User Interface Kontroll- oder Eingabeelemente.

Das Android-Betriebssystem wurde unter der Open Source Apache-Lizenz veröffentlicht und setzt auf einen Linux Kernel v2.6 auf. Android ist ein Produkt der von Google gegründeten „Open Handset Alliance“ (OHA), welche mittlerweile 65 Hard- und Software Unternehmen vereint. [Allen 2010, S. 35] Im Gegensatz zur Arbeit mit iOS verfügt ein Programmierer auf der Google-Plattform über mehr Freiheiten. Die für Android empfohlene Entwicklungsumgebung Eclipse IDE und das benötigte Android Development Tool Plug-In sind unter Windows, Linux sowie MacOS lauffähig. Applikationen werden typischerweise in Java geschrieben, wobei die Java-Klassen jedoch in Ermangelung einer Java Virtual Machine in Dalvik-Bytecode rekompiliert und auf einer Dalvik Virtual Machine ausgeführt werden [Allen 2010, S. 36].

■ DALVIK: Open-source Software mit registerbasierter Architektur.

„Windows Phone 7"-Anwendungen werden primär unter Verwendung der Microsoft Tools „Visual Studio 2010" und „Expression Blend 4" entwickelt, die ausschließlich unter dem Betriebssystem Windows lauffähig sind. Auch „Microsoft Silverlight" und das „XNA Game Studio" Framework eignen sich für die App- und Spiele-Entwicklung. [Allen 2010, S. 66] Programmiert werden „Windows Phone 7" Apps in der von Microsoft entwickelten ECMA- und ISO-normierten Sprache C#.

Das Blackberry Tablet OS unterstützt mehrere Technologien zur Erstellung von Apps. Zum einen können Entwickler über Blackberrys neue Plattform WebWorks mit Hilfe von Standard Webtechnologien wie CSS, HTML5 und JavaScript Web-Applikationen mit Hardwarezugriff schreiben, zum anderen lassen sich native Apps mit dem Adobe AIR SDK umsetzen. Für die Entwicklung empfiehlt RIM die Verwendung von Adobe Flash Builder 4.5 und bietet einen Emulator für Windows, Mac und Linux an, der durch VMWare realisiert wird. Programmiert werden Adobe AIR Apps in ActionScript 3 und MXML.

- SILVERLIGHT: Framework von Microsoft zum Erzeugen und Ausführen von RIA-Anwendungen.
- XNA: Game Studio Framework von Microsoft.
- ECMA: „European Computer Manufacturers Association"
- ISO: Internationale Organisation für Normung.
- VMWARE: Software zum Virtualisieren von Betriebssystemen.

II.2.4.2 Distribution nativer Anwendungen

Bevor mit der Entwicklung für iOS gestartet werden kann, ist die Registrierung im Apple Developer Program erforderlich. Dies beinhaltet einen kostenpflichtigen Developer-Account, der jährlich für 99 US Dollar erneuert werden muss. Über ein aufwendiges Zertifizierungsverfahren müssen alle für die Entwicklung benötigten Device-IDs erfasst werden. Dadurch wird das Testen ausschließlich auf die registrierten Geräte beschränkt.

Bei Android gestaltet sich die Distribution weniger umständlich, da weder eine Registrierung in einem Developer Programm notwendig ist, noch eine Signierung der Applikation. Erst für die tatsächliche Veröffentlichung im Android Market ist eine Gebühr von 25 US Dollar zu entrichten und eine digitale Signatur mittels selbst erstelltem Private Key vorzunehmen. [Allen 2010, S.49f] Live-Debugging auf beliebigen Testgeräten ist sofort nach dem Download des SDKs möglich.

Für „Windows Phone 7" ist das Programmieren und Testen von Apps mit Hilfe eines Emulators ähnlich wie bei Apple kostenfrei. Ausführen des

Codes auf einem realen Gerät sowie die Distribution über den angebundenen Market verlangen jedoch eine Einmalzahlung von 99 US Dollar sowie die Registrierung jedes Entwicklungsgerätes. [Knor 2011, S.37]

Die Anmeldung einer RIM App für die Blackberry App World dagegen ist kostenlos. Nach der Erstellung eines Kreditorenkontos kann die Anwendung unter Beachtung der BlackBerry App World™ Vendor Guidelines[1] sowie erfolgreicher Signierung eingestellt werden.

iPhone und iPad Apps können ausschließlich über Apples App-Store veröffentlicht und bezogen werden. Die Distribution einer Anwendung unterstellt Apple strengen Kriterien[2], wobei sich das Unternehmen vorbehält, jede App einzeln zu prüfen und gegebenenfalls abzulehnen. Im Kontrast zu diesem langwierigem Prozess erlaubt Android das direkte Einstellen von Apps im offiziellen Google Android Market. Applikationen unterliegen lediglich den „Android Market Developer Distribution Agreement"[3], bedürfen aber keiner separaten Prüfung. Zudem existieren, anders als bei der monopolistischen Stellung des Apple-Stores, für Android zahlreiche alternative Vertriebskanäle wie zum Beispiel der bereits in den USA gestartete App-Market von Amazon®, die Markets „AppBrain" und „AndroidPIT", sowie die Bereitstellung von Apps zum Download oder manueller Installation.

■ WP7: Kurzform für „Windows Phone 7"

Um WP7 Apps öffentlich zugänglich zu machen, werden diese über die App-Hub-Seite für den Windows Phone Marketplace publiziert. Über die unabhängige Firma „Geo-Trust" muss zunächst die eigene Identität bestätigt werden. Anschließend wird die Anwendung auf die Richtlinien für Windows-Phone-7 Applikationen[4] geprüft. Dieser Prozess dauert etwa eine Woche und erfolgt „teils automatisch, teils manuell". [Knor 2011, S.46]

Zusammenfassend kann festgehalten werden, dass native Applikationen für jedes der beschriebenen Systeme in einer anderen Programmiersprache, in einer anderen IDE, unter Verwendung unterschiedlicher SDKs und Emulatoren erstellt werden müssen. Dieser Aufwand kann durch den Einsatz einer Cross-Plattform-Lösung erheblich reduziert werden. Unüber-

1 https://appworld.blackberry.com/isvportal/home/guidelines.seam
2 http://developer.apple.com/appstore/guidelines.html
3 http://www.android.com/us/developer-distribution-agreement.html
4 http://go.microsoft.com/?linkid=9730558

windbar scheint jedoch die Bewältigung der nötigen Schritte und Kosten zur Registrierung der Anwendung in den jeweiligen Vertriebsmärkten zu bleiben.

II.2.4.3 Unterstützung von Flash und AIR

Die größte Unterstützung findet das Flash Player Plugin v10.2 bisher auf Geräten mit dem Android OS ab v2.2. Auch die Adobe Integrated Runtime v2.6 ist seit Ende Februar 2011 für Android Geräte der Versionen 2.2, 2.3 und Honeycomb (3.0) verfügbar. Im gleichen Maße integriert das Blackberry Tablet OS das neueste Flash Player Plugin sowie Adobe AIR.

Außenseiter bleibt Apples iOS. Wie bereits erwähnt wird Apples iOS trotz massiver Fortschritte Adobes in der Anpassung des Players für mobile Geräte auch weiterhin keine Flash-Inhalte im Browser darstellen können. Anders stellt sich jedoch die Unterstützung von Adobe AIR dar. Mit dem Update des Adobe Flash Builders auf Version 4.5 im April 2011 lassen sich ab sofort mobile Applikationen für Android, Blackberry Tablet OS und iOS erstellen. Auch für Windows Phone 7 wurde die Flash Player Integration angekündigt [Chambers 2010]. Zur Bereitstellung von Adobe AIR für WP7 gibt es noch keine konkreten Aussagen.

II.3 Cross-Plattform Strategien im Vergleich

Um mit einer individuellen Applikation die Mehrzahl aller Mobilgeräte-Nutzer zu erreichen muss heute entweder für jede Plattform eine seperate native App entwickelt werden oder man bedient sich einer Cross-Plattform-Strategie, um die Kosten gering zu halten [Behrens 2010]. Die meisten Unternehmen können es sich jedoch nicht leisten, ein einziges Gerät zu favorisieren, da sich ihre Kunden über alle Systeme verteilen. Solche extremen Herausforderungen in der plattformübergreifenden Entwicklung begünstigen die Bildung von Cross-Plattform-Frameworks. [Allen 2010, S.5f]

Nach Behrens [Behrens 2010] lassen sich vier Ansätze in der plattformübergreifenden App-Entwicklung unterscheiden: Web Apps, Hybride Apps, Interpretierte Apps und Generierte Apps. Im Folgenden werden diese kurz erörtert und anschließend eine Einordnung der Adobe Integrated Runtime vorgenommen. Vergleichsgrundlage bilden die Kriterien Entwicklungsaufwand und Zugriffsmöglichkeiten auf native Plattformfunktionen.

II.3.1 Strategien

II.3.1.1 Web Apps

Das mobile Netz ist die einzige Plattform, die systemübergreifend verfügbar ist und für alle Geräte dieselben Standards und Protokolle bereitstellt [Fling 2009, S.145]. Mit einer Website erreicht man quasi jedes mobile Endgerät. Diesen Vorteil nutzend, entstehen zahlreiche Webseiten, die spezielle, an kleine Displays und touchbasierte Eingaben angepasste UI-Elemente einsetzen.

Web-Frameworks, wie iWebKit, iUI, JQuery mobile und Sencha Touch, unterstützen den Programmierer dabei, mit Hilfe von Web-Technologien, wie HTML, CSS und Javascript nativ-wirkende Interfaces zu erstellen, die im Webbrowser des Mobiltelefones laufen.

Sencha Touch[5] beispielsweise ist ein auf HTML5, CSS3 und Javascript basierendes Framework, welches unter Bereitstellung eigener Datenpakete und integrierter UI-Elemente die Erstellung einheitlich aussehender Web Apps für iPhone, Android und Blackberry ermöglicht. Es wurde unter der open source license GPL v3 veröffentlicht und bezeichnet sich als das erste mobile HTML5 Web App Framework. Ein großer Vorteil des Web-App-Ansatzes ist die Verwendung bekannter Technologien, sowie vorhandener Prozesse und Tools. Sie können in kurzer Zeit entwickelt werden und Softwareupdates sind somit leichter und transparenter zu bewerkstelligen. Nachteilig wirkt sich die limitierte Hardwareunterstützung aus, sowie die Performance-Probleme der Browser-Engines. Auch lassen sich die meisten Gerätefunktionen nicht integrieren. Geolocation und Offline-Verfügbarkeit sind jedoch generell möglich. Da Web-Apps im Browser des mobilen Endgerätes laufen und nicht direkt installiert werden, stellen sie in dem Sinne keine „echten", nativen Apps dar. Aus diesem Grund lassen sie sich nicht über einen App-Store vertreiben, wodurch das Marketing solcher Apps deutlich erschwert wird.

II.3.1.2 Hybride Apps

Technologisch gesehen versteht man unter einer hybriden Applikation einen schmalen nativen Container, der einzig aus einem eingebetteten Webbrowser besteht. Die eigentlichen Inhalte werden in diesem Browser als Website unter Verwendung bekannter Standards, wie HTML, CSS und Javascript, dargestellt. Somit heben hybride Apps den Nachteil des fehlenden Vertriebskanals bei webbasierten Apps auf, da sie auf die herkömmliche Weise über die verschiedenen Markets verteilt werden können. Dank der nativen Hülle können mittels einer Javascript-API zusätzliche Gerätefunktionen, wie Telefonereignisse, Vibration, GPS-Position, Kamera- und Beschleunigungsfunktionen, angesprochen werden.

PhoneGap[6] ist eines der ersten freien Frameworks, mit dem sich mobile Web Applikationen in native Anwendungen umwandeln lassen. Die Software unterstützt bisher sechs verschiedene Plattformen, zu denen unter

5 http://www.sencha.com/products/touch
6 http://www.phonegap.com

anderem iOS, Android und Blackberry zählen. Als HTML5-App-Plattform erlaubt es Entwicklern, native Applikationen mit Web-Technologien zu verbinden und stellt Zugänge zu System-APIs und App Markets bereit. Die Projekte werden für jedes System getrennt angelegt und unter Verwendung der nativen SDKs kompiliert. Für die Entwicklung einer iPhone App beispielsweise muss das iPhone SDK installiert sein und eine Registrierung beim Apple Developer Programm vorgenommen werden. Dennoch gibt es keine Gewissheit, dass eine hybride PhoneGap Anwendung im App Store akzeptiert wird.

Die Arbeit mit Hybriden bietet einige Vorteile. User Experience, Offline Verfügbarkeit und Hardwareunterstützung verhalten sich jedoch wie im webbasierten Kontext, so dass die volle Bandbreite eines nativen Nutzererlebnisses nicht abbildbar ist.

II.3.1.3 Interpretierte Apps

Mit dem Begriff der interpretierten Applikationen nimmt Behrens eine klare Abgrenzung zu den Hybriden vor. Bei diesem Ansatz wird nicht mehr auf Web-Standards zurückgegriffen und statt des Browsers wird eine bestimmte Laufzeitumgebung als Interpretationsinstanz verwendet. Interpretierte Apps verwenden eigene oder plattformspezifische UI-Elemente, wodurch ein natives look-and-feel möglich wird. Die Funktionslogik wird in einer unabhängigen Sprache implementiert und kann anschließend für die gewünschte Zielplattform exportiert werden.

Einer der bedeutendsten Vertreter ist das Framework Appcelerator Titanium[7], welches ähnlich wie das zuvor erwähnte PhoneGap auf eine Javascript-API setzt, ohne dabei HTML und CSS zu benötigen. Dabei interpretiert eine im Hintergrund verankerte WebView den Javascript Inhalt und kommuniziert mit dem Wirtssystem. „Titanium Mobile stellt in der Host-Sprache entwickelte Module zur Verfügung, die [...] zur Laufzeit aufgerufen werden und native Elemente erzeugen“ [Leber 2011, S.66]. Titanium ist ein unter der Apache 2 Lizenz veröffentlichtes open-source Framework, welches mit Hilfe von Wrappern die direkte Integration von Twitter, Facebook,

- LOOK-AND-FEEL: Anmutung und Handhabung von Design und User-Interface.
- Wrapper: Bezeichnung für ein Stück Software, welches ein anderes Stück Software umgibt
- TWITTER: Plattform zum Verbreiten von Kurznachrichten im Internet.
- FACEBOOK: Soziales Netzwerk mit über 700 Mio. Mitgliedern weltweit.
- RSS: „Really Simple Syndication” - XML-basiertes Internet-Nachrichtenformat.
- SOAP: Netzwerkprotokoll zum Austausch von Daten zwischen Systemen über HTTP und TCP.
- SOCKET: Dient der bidirektionalen Kommunikationsverbindung zwischen zwei Programmen.

7 http://www.appcelerator.com

RSS und SOAP-APIs ermöglicht, wie auch den Zugang zu Sockets, HTTP-Verbindungen, Filesystem und lokaler Datenbankspeicherung [Allen 2010, S.158]. Andere Vertreter, wie MonoTouch (.NET) oder Rhodes Rhomobile (Ruby) basieren auf komplett eigenen, voll funktionstüchtigen Laufzeitumgebungen.

■ .NET: Entwicklungsplattform von Microsoft.

Der Interpretationsansatz bietet einige Vorteile, wie den Zugang zur Hardware, Offline-Verfügbarkeit, Monetisierung über einen App Market und eine native User-Experience. Gleichzeitig sind Entwickler jedoch an das gewählte Framework und dessen Paradigmen gebunden. Neuerungen in den jeweiligen System-APIs müssen zunächst von den Third-Party-Tools integriert werden, bevor sie genutzt werden können. Gegebenenfalls müssen neue Programmiermodelle und -schnittstellen für das gewählte Framework erlernt werden, wofür es im Vergleich zu den originalen Entwicklungssprachen meist weniger Support gibt. Dennoch erlaubt diese Methode gegenüber den webbasierten Ansätzen Zugriff auf weitaus mehr Gerätefunktionen. Dazu zählen: GPS, Vibration, Accelerator, Sound, Fotoalbum (Betrachten und Speichern), Ausrichtung, Kamera, Screenshots, Videoaufnahmen, Kontakte und Kalender [Allen 2010, S.157].

II.3.1.4 Generierte Apps

Eine weitere Möglichkeit plattformübergreifende Anwendungen zu entwickeln liegt darin, einen eigenen Generator zu entwerfen, welcher den in einer beliebigen Sprache programmierten Code in den Code der Zielplattform übersetzt. Dadurch entstehen vollständig native Applikationen, die auf jedes einzelne Feature der Host-Plattform zugreifen können. Mit dem XMLVM-Cross-Compiler[8] beispielsweise kann der in Java programmierte Basiscode in Javascript, .NET und Objective-C kompiliert werden. Der erzeugte Programmcode ist leider kaum noch lesbar. Mit anderen Lösungen, die auf einer domain-specific language (DSL) basieren, lässt sich dieser Nachteil vermeiden. „Die Erstellung und Pflege eines Generators und einer DSL […] kann recht aufwändig werden und verlangt ein gewisses Spezial-Know-how für den Einstieg“ [Leber 2011, S.70] Kann diese Hürde über-

8 http://xmlvm.org

wunden werden, stehen dem Entwickler mit dem generativen Ansatz alle Vorteile einer nativen Anwendung zur Verfügung. Aktuell ist jedoch noch keines der derzeit öffentlich verfügbaren Werkzeuge in der Wirtschaft einsatzbereit, so dass dieser Ansatz eher als ein theoretischer zu betrachten ist.

II.3.1.5 Zusammenfassung

Trotz der Vielzahl an Cross-Plattform Werkzeugen zeigt Behrens auf, dass bisher keine der Lösungen uneingeschränkt praxistauglich ist. Die Tabelle 1.1 veranschaulicht diesen Aspekt. Web- und Hybride Apps lassen sich mit minimalem Aufwand entwickeln, müssen jedoch Abstriche in der User-Experience und Funktionsunterstützung hinnehmen. Frameworks, die die Integration nativer Features erlauben, erfordern wiederum hohe Einarbeitungszeiten in entsprechende APIs.

	web	hybrid	interpreted	generated
Entwicklung				
Aufwand				
App Market				
Experience				
Hardware				
Offline				

Tabelle 1.1: Bewertung der Cross-Plattform Strategien in Anlehnung an [Behrens 2010]

II.3.2 Einordnung Adobe AIR

Ähnlich wie MonoTouch und Rhomobile verwendet Adobe AIR eine eigene Virtual Machine, welche den aus ActionScript kompilierten Bytecode ausführt. Folglich lassen sich mit dem AIR Framework erstellte Applikationen dem interpretativen Ansatz zuordnen. Auch AIR stellt eigene UI-Elemente für ein natives Aussehen zur Verfügung. Die Funktionslogik wird in ActionScript 3 und wahlweise MXML unter Verwendung des Flex-SDKs implementiert. Mit der aktuellen Flash Builder Version 4.5 lassen sich aus einem Projekt Applikationen für iOS, Android und das Blackberry Tablet OS erstellen. Die Zuordnung zum interpretativen Ansatz gilt nur, soweit die Adobe Runtime auf dem mobilen Endgerät benötigt wird, um AIR Apps auszuführen. Dies ist für Android und Blackberry Tablet OS der Fall. Eine Ausnahme bildet jedoch die Einordnung der für iOS kompilierten AIR Apps.

Laut der aktuellen Richtlinien für die Publikation von Apps für iOS ist es nicht möglich, die Adobe Runtime auf einem iPhone/ iPad zu installieren oder sie in das Export-Package der eigentlichen App zu integrieren.

> „Apps that install or launch other executable code will be rejected“
> [Apple 2011]

Aus diesem Grund verwendet Adobe einen LLVM-Compiler, welcher die Runtime zusammen mit dem ActionScript Quellcode in nativen ARM Assembler-Code für iOS kompiliert. So wird für das finale Binärfile keine Laufzeitumgebung mehr benötigt und die generierte App ist hundertprozentig nativ. [Bansod 2011] In diesem besonderen Fall greift Adobe auf eine generative Strategie zur App-Erstellung zurück.

- LLVM: „Low Level Virtual Machine “ - Eine modularer Bytecode Compiler und eine Virtual Machine.
- ARM: „Advanced RISC Machines”- Chip Architektur.

Die Adobe Runtime vereint alle Vorteile des interpretativen Ansatzes: geringer Entwicklungsaufwand, Monetisierung und Offline-Verfügbarkeit. Bei bereits vorhandenen Kenntnissen von ActionScript und Flex verschwinden auch die Nachteile eines eventuellen Einarbeitungsaufwands. Entwicklungssupport ist durch das langjährige Zusammenwirken der Flashgemeinde gewährleistet.

III Konzept und Technische Realisierung des Prototyps

III.1 Konzept der App „Taskmanager“

III.1.1 Anforderungen

Ziel der Applikation „Taskmanager“ soll es sein, den Mitarbeitern von „Zeros+Ones“ das tägliche Erfassen der Arbeitszeit pro Kundenprojekt zu erleichtern, welches momentan über eine Intranetseite erfolgt. Dies wird vor allem durch eine größere Erreichbarkeit des aktuellen Systems realisiert. So soll die App am Arbeitsplatz, im Home Office sowie unterwegs einen Zugang zum internen System der Agentur ermöglichen. Der bisherige Funktionsumfang wird um neue Features erweitert, wobei das automatische Mitloggen der Arbeitszeit einzelner Projekte eine der größten Neuerungen darstellt. Neue Projekte werden wie gewohnt über die Software „FileMaker“ angelegt und können anschließend durch einen Projektleiter über den Taskmanager gepflegt werden.

Technisch betrachtet soll der Taskmanager Browser- und Systemunabhängig funktionieren. Dabei werden konkrete Schnittstellen zum Server definiert, die von beliebigen Clients angesprochen werden können. Der Taskmanager soll sowohl als Desktop-Widget wie auch als mobile Applikation einsetzbar sein. Voraussetzung für den erfolgreichen Einsatz ist in jedem Fall eine bestehende Internetverbindung.

Da die Applikation einen Dienst mit hoher User-Interaktivität bereitstellt, lässt sie sich konzeptionell dem im Kapitel II.2.2 vorgestelltem „Productivity Context“ zuordnen. Auf verschiedenen Screens soll der User in der Lage sein, Daten einzugeben und Zuweisungen zu tätigen. Das bisherige Vorgehen soll durch das neue Modell in ein einfaches Bedienkonzept überführt werden.

Funktionen

Der Start in die Anwendung erfolgt für den Mitarbeiter über die Eingabe des persönlichen Nutzerkennzeichens und Passwortes. Nach der erfolgrei-

chen Prüfung der Zugangsdaten wird der Start-Screen in den Fokus navigiert. Abhängig von der im System zugeordneten Rolle, werden für jeden Mitarbeiter unterschiedliche Funktionen geladen. Aktuell sind die Rollen „Mitarbeiter“ und „Projektleiter“ vorgesehen.

Nach dem Login besteht die Benutzeroberfläche der Anwendung aus drei Hauptseiten-Elementen. Die „Aktive Zeiterfassung“ bildet den Einstiegspunkt in die App. Die anderen beiden Navigationspunkte „Tagesansicht“ und „Filtern“ sind parallel anwählbar. Über ein globales Menü können aus jeder der drei Views heraus Datum und Modus geändert, sowie die Suche aufgerufen werden.

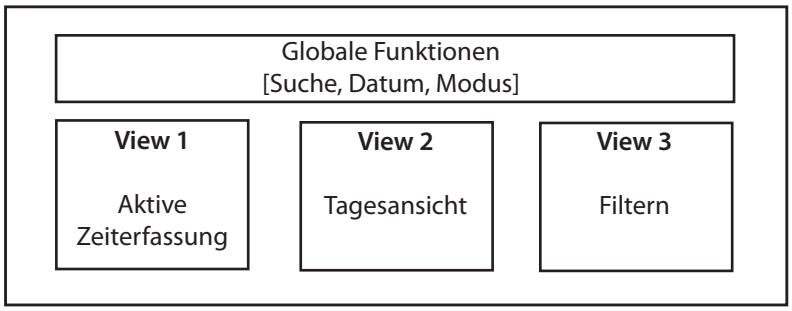


Abbildung 3.1: Aufbau der Hauptseiten-Navigation des Taskmanagers

Dieses viewübergreifende Menü zeigt weiterhin den Namen des eingeloggten Mitarbeiters sowie die seit dem Login verstrichene Zeit an. Sind im System noch ausstehende Tage vorhanden, die vom Mitarbeiter nachgetragen werden müssen, erscheint unter dem Datum ein Hinweistext. Über die Moduswahl kann dem jeweiligen Tag einer der folgenden Modi zugewiesen werden: normal, krank, frei, Urlaub oder extern.

Ein essentieller Bestandteil der App ist die zentrale Suche. Sie ermöglicht das Finden und Hinzufügen von Projekten über die Eingabe der Projektnummer oder des Projektnamens. Beim Tippen des Suchbegriffs werden in einer Liste automatisch zur Eingabe passende Ergebnisse vorgeschlagen. Treffer, die bereits in der Projektliste des Mitarbeiters vorhanden sind, werden in der Ergebnisliste zuerst angezeigt, weitere Treffer darunter. Durch die Auswahl eines Eintrages in der Trefferliste wird dieses Projekt der eigenen Liste hinzugefügt oder in den Fokus geholt, falls es bereits zugewiesen war.

Als ein besonderes Feature wird über dem globalen Menü eine Art Progressanzeige integriert. Der dargestellte Fortschritt repräsentiert das Level welches der Mitarbeiter durch zeitnahes und effizientes Buchen seiner Stunden bereits erreicht hat. Durch diese Funktion sollen die Mitabeiter zum rechtzeitigen Kontieren ihrer Projekte motiviert werden.

View 1 – Aktive Zeiterfassung

Die Konzept-Mockups (Abbildung 3.2) zeigen die Struktur der Aktiven Zeiterfassung. Primäres Element des Haupt-Screens ist die Projektliste, welche alle dem Mitarbeiter in der Datenbank zugeordneten Projekte beinhaltet. Von hier aus kann für ein beliebiges Projekt die Arbeitszeit mittels Start/ Stop-Button automatisch erfasst werden. Über der Projektliste befinden sich Buttons für die zwei Kernfunktionen „Telefon“ und „Pause“. Wie auch für Projekte kann für beide die aufgewendete Zeit mitgeloggt werden. Da es sich beim Telefonieren und Pausieren um Tätigkeiten handelt, die die normale Projektarbeit öfter unterbrechen können, sind diese außerhalb der Projektliste untergebracht. Wurde für ein Projekt die Zeiterfassung über den Stop-Button beendet, wird dieses nach der Bestätigung eines Dialogs in die Tagesansicht übernommen und kann dort nachbearbeitet werden. Dies gilt ebenso für Telefonate und Pausen, wobei Telefonate im Bestätigungsdialog einem konkreten Projekt bzw. Kunden zugeordnet werden müssen.

■ MOCKUP: Entwurfsmodell der Bedienoberfläche und Interaktionsverzweigungen.

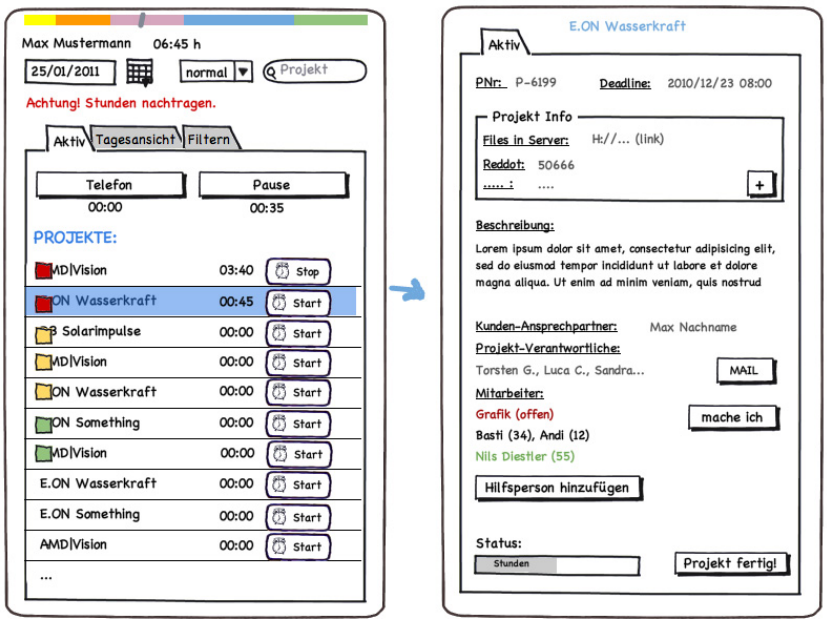


Abbildung 3.2: Konzept-Mockups für die „Aktive Zeiterfassung“ der Taskmanager-App, links: Projektliste, rechts: Projektdetail-Ansicht

Nach der Auswahl eines Projekteintrags in der Favoritenliste sollen dessen Projektdetails auf einem neuen Screen anzeigt werden. Die Details informieren den Anwender unter anderem über Projektnummer, Projekt-Deadline, Serverdaten, Abarbeitungsstatus, Ansprechpartner und ebenfalls am Projekt arbeitende Mitarbeiter. Eine Email-Funktion ermöglicht es den Projektverantwortlichen auf schnellem Weg eine Nachricht zu senden. Dem User stehen hier abhängig von seiner Rolle verschiedene Funktionen zur Verfügung. Mitarbeiter haben die Möglichkeit, eine Hilfsperson zum Projekt hinzuzufügen. User in der Rolle „Projektleiter" können sämtliche Informationspunkte bearbeiten. Dies betrifft zum Beispiel das Hinzufügen und Löschen von Mitarbeitern, Ansprechpartnern, Projektverantwortlichen und Unterprojekten.

View 2 – Tagesansicht

Die Tagesübersicht ist neben der aktiven Zeiterfassung der bedeutendste Bestandteil der Applikation. Hier werden für den jeweiligen Tag sämtliche gebuchte Arbeitszeiten dargestellt. Dabei werden die eingetragenen Zeitblöcke pro Projektnummer gruppiert und deren Zeiten summiert. So können zu einem Projekt mehrere Projekt-Events angelegt werden, zu denen auch Telefonate zählen. Für jedes einzelne Projekt-Event muss der Projektname, ein Ansprechpartner, die Beschreibung der Tätigkeit sowie die Bearbeitungsdauer angegeben werden.

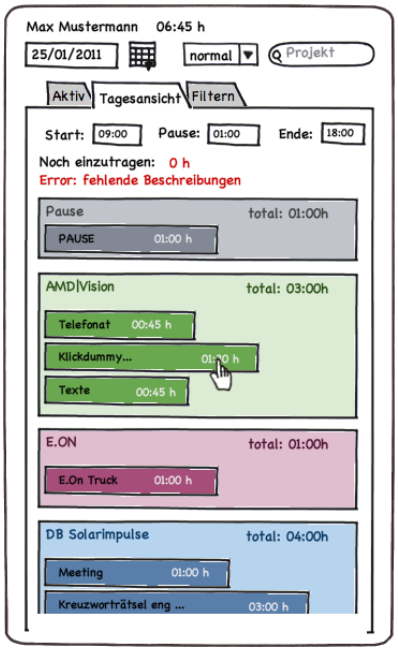


Abbildung 3.3: Konzept-Mockups für die „Tagesansicht" der Taskmanager-App

Wurde die Zeit nicht automatisch mittels Start/Stop-Funktion erfasst, kann ein Projekt auch manuell zur Tagesansicht hinzugefügt werden. Dies kann auf zwei Wegen erreicht werden. Zum einen kann der User bei aktivierter Tagesansicht über die globale Suche Projekte finden und hinzufügen. Zum anderen gibt es in der Projektliste der aktiven Zeiterfassung für jeden Eintrag einen Button „zur Tagesansicht hinzu-

fügen". Für manuell hinzugefügte Projekt-Events müssen anschließend die Details nachgetragen werden.

Für jeden Tag können Start- und Endzeitpunkt festgelegt werden sowie die Dauer der Pause. Sind noch Stunden nachzutragen oder fehlt einem Event der Beschreibungstext, wird der User vom System darauf hingewiesen.

View 3 – Filtern

User in der Rolle „Mitarbeiter" können sich auf dem dritten Haupt-Screen die eigene Auslastung über alle Projekte pro Tag, Woche und Monat anzeigen lassen. Für Projektleiter dagegen zeigt die View zusätzlich die Auslastung einzelner Mitarbeiter wie auch den Gesamtstundenverbrauch von Projekten. Zudem kann sich ein Projektleiter eine Übersicht erstellen lassen, welcher Mitarbeiter an welchem Projekt arbeitet und somit freie Kapazitäten erkennen. Die Filter-Ansicht soll in einer späteren Entwicklungsphase umgesetzt werden und wird für den Prototyp noch keine Rolle spielen.

III.1.2 Design- und Usabilityvorgaben

Auf Grundlage der Mockups (vgl. Anhang A.1) wurde durch die Grafikabteilung der Agentur „Zeros + Ones“ ein Design- und Usabilitykonzept (vgl. Anhang A.2) erstellt. Die Farb- und Formgebung wurde dabei an das bisherige CI/CD des Intranets angepasst. Akzentuierende Farben sind Magenta und Cyan. Eine planare Gestaltung der Interaktionselemente sorgt für ein aufgeräumtes und klar strukturiertes User-Interface. Hierbei bestand das Ziel vorrangig darin, ein komplett losgelöstes Design zu schaffen, welches weder an iOS noch an andere Mobilsysteme erinnert.

Usability – Mobile App

Beim Entwurf des Usabilitykonzepts der mobilen App wurden sämtliche Interaktionsvorgaben zunächst für die Nutzung eines Multitouch-Displays ausgelegt. Für das, aus der selben Codebasis hervorgehende, Desktop-Widget werden davon abweichende Interaktionen definiert, wobei Design und Navigationsprinzip übernommen werden. Auf die Abweichungen wird am Ende des Kapitels kurz eingegangen.

Zwingend für eine erfolgreiche Nutzung mobiler Applikationen sind ausreichend große Kontrollelemente. Buttons und Textfelder sollten eine Mindesthöhe von 7mm/26px haben [Wroblewski 2010], welches dem Touchpoint eines durchschnittlichen Fingers entspricht. Auf Multitouch-Displays werden alle Selektionen, die auf dem Desktop mittels Mausklick vorgenommen werden, über einen Single-Touch (Tap) umgesetzt.

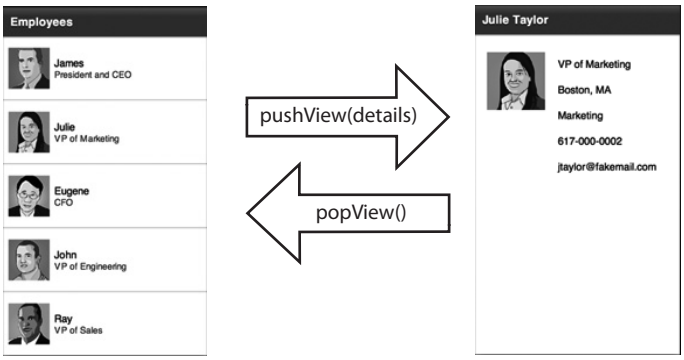


Abbildung 3.4: Screen Metapher Prinzip, angelehnt an ein Projekt von Michaël CHAIZE (http://www.riagora.com/sfdc/sfdcemp.fxp.zip)

Auch die Inhalte einer App müssen für die kleinen Displays aufbereitet werden. Im Gegensatz zum Desktop verfügen Smartphone Screens über weitaus weniger Platz, um Informationen darzustellen. In diesem Zusammenhang hat sich für Mobilgeräte das Konzept der sogenannten „Screen Metapher" durchgesetzt. Dabei werden die Informationen der Applikation auf eine Reihe von verschiedene Screens, auch Views genannt, aufgesplittet. Sollen beispielsweise die Details eines bestimmten Listenelementes der aktuellen View zur Anzeige kommen, werden diese auf einem neuen Screen präsentiert (vgl. Abbildung 3.4). Will der Anwender wieder zurück zur Übersicht, wird der vorherige Screen in den Fokus geholt. [Corlan 2011]

Für das Einblenden eines neuen Screens haben sich verschiedene Übergangstypen durchgesetzt, die sich auf den einzelnen Mobilsystemen sehr ähneln. Am häufigsten wird der lineare Übergang eines Screens von rechts nach links verwendet. Hierbei schiebt sich der neue Screen quasi von rechts ins Bild und verdrängt dabei den aktuellen Screen nach links aus dem Bildbereich. Dieser Übergangstyp wird auch für die Haupt-Screen-Navigation der Taskmanager-App verwendet. Um den Einsatz von untergeordneten Screens wie beispielsweise Kalender und Suche zu visualisieren, werden diese mittels einer linearen Transition von unten nach oben in den Fokus navigiert. Dadurch sollen die verschiedenen Navigationsachsen besser veranschaulicht werden.

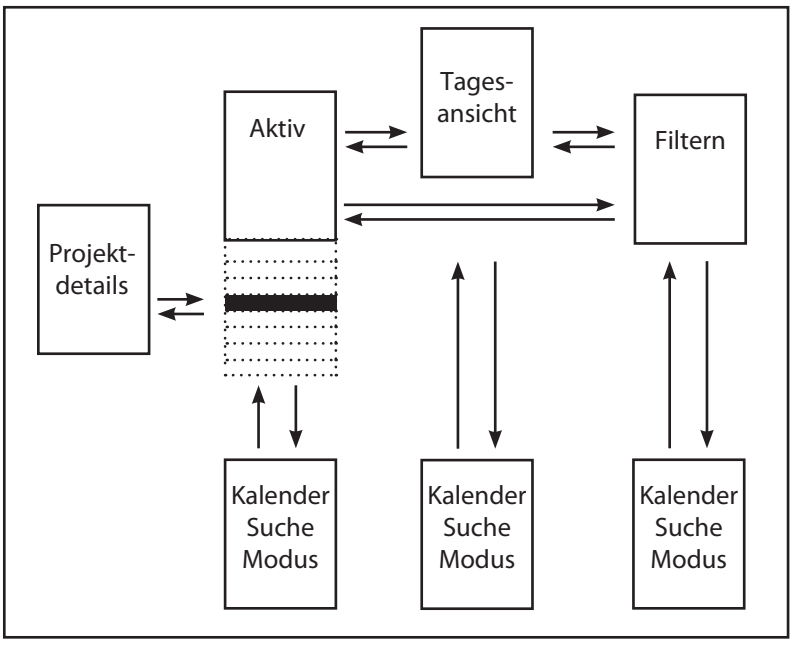


Abbildung 3.5: Vertikale und horizontale Übergänge zwischen den Haupt-Views des Taskmanagers

Für das Einstellen von Zeitangaben in der Tagesansicht soll statt des integrierten Keyboards ein eigener Slider verwendet werden, welcher auf einem extra Screen von unten nach oben eingefadet wird. Der Slider für die Zeitwahl beispielsweise besteht aus zwei vertikalen Touchbereichen, wobei der linke Bereich die Stunden anzeigt und der Rechte die Minuten. Über langsames Wischen eines Fingers entlang der vertikalen Achse eines der beiden Bereiche werden die Zahlen entsprechend der Wischrichtung verschoben. Auf diese Weise lassen sich Zeitangaben intuitiver anpassen und Fehlerquellen bei der Eingabe über Tastatur vermeiden.

Abbildung 3.6: Designentwurf der Zeitslider-View des Taskmanagers

Auch versteckte Funktionen lassen sich bequem mittels Multitouch-Gesten integrieren. So verfügt beispielsweise die Projektliste eines Mitarbeiters in der „Aktiven Zeiterfassung“ auf den ersten Blick über keinerlei Funktionalitäten. Der Nutzer kann jedoch unter Anwendung einer einfachen Wisch-Geste von links nach rechts auf einem Listeneintrag zusätzliche Buttons zum Vorschein bringen. Diese werden auf dem jeweiligen Eintrag eingeblendet und über dieselbe Wisch-Geste in umgekehrter Richtung wieder ausgeblendet. Diese Art von Multitouch-Gesten werden allgemein auch als Swipe-Gesture bezeichnet und üblicherweise zum Einblenden von Bearbeitungsmöglichkeiten, wie Löschen oder Verschieben von Listeneinträgen, verwendet. Im Fall des Taskmanagers dienen die zwei eingeblendeten Buttons dazu, das ausgewählte Projekt zur Tagesansicht hinzuzufügen oder das Loggen der Arbeitszeit zu starten.

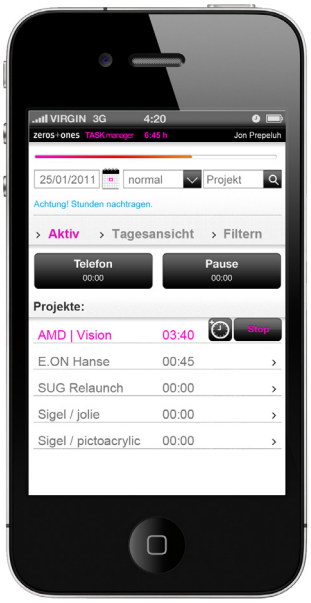


Abbildung 3.7: Designentwurf der Aktiv-View des Taskmanagers

Auf dem Screen „Tagesansicht“ des Taskmanagers werden einzelne Projekt-Events als Balken visualisiert. Die Länge eines Balkens repräsentiert dabei die Dauer des gespeicherten Eintrags. Mittels touch-basiertem Drag&Drop sollen die Balken in ihrer Länge schrittweise in einem Raster von 15 Minuten minimiert und maximiert werden können. Über einzelne Taps auf einen Balken, kann der User einen Projekteintrag löschen oder manuell nachbearbeiten.

Usability – Desktop-Widget

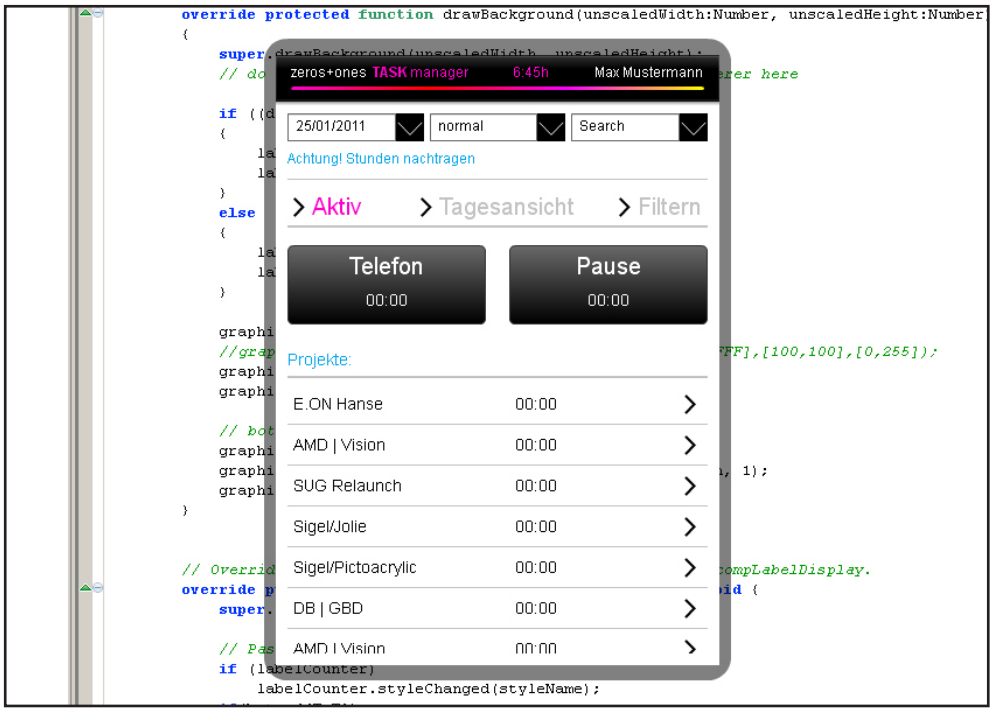


Abbildung 3.8: Desktop-Widget Version des Taskmanagers

Aus der Codebasis der Taskmanager-App soll zusätzlich eine AIR-Anwendung für den Desktop kompiliert werden. Da auf Desktops die Interaktion zu meist noch mittels Computer-Maus erfolgt, können hier die bei der mobilen App eingesetzten Multitouch-Interaktionen nicht verwendet werden. Es ist jedoch nicht nötig alle Events abzuändern. Normale Single-Taps sollen auch auf dem Desktop wie herkömmliche Klicks mit der linken Maustaste behandelt werden. Auch das Scrollen in Listen lässt sich äquivalent zur Touch-Variante umsetzten, indem man die Maus bei gedrückter linker Maustaste über der Liste hoch und runter bewegt. Zusätzlich kann der Anwender auch mittels Mausrad in der Liste scrollen. Das selbe Prinzip kommt beim Zeitslider zum Einsatz. Statt eines Fingers wird die linke Maus-

taste zum Starten des Scrollvorgangs verwendet. Auch die durch Balken visualisierten Projekt-Events der Tagesansicht lassen sich beim Desktop-Widget durch normales Drag&Drop mittels Maus verlängern oder verkürzen.

Eine größere Abweichung zwischen App und Widget gibt es in der Interaktion mit der Projektliste. Mit Hilfe einer Wisch-Geste von links nach rechts auf einem Listeneintrag können in der App versteckte Buttons zum Vorschein gebracht werden. Diese Geste mittels Maus für den Desktop nachzubilden ist weniger sinnvoll, da sie aus Usabilitysicht für den Anwender nicht intuitiv nutzbar sein wird. Stattdessen hat sich auf dem Desktop seit langer Zeit der rechte Mausklick zum Hervorbringen eines Kontextmenüs durchgesetzt. Im Kontextmenü werden für das jeweilige Objekt, auf welchem der rechte Mausklick ausgeführt wurde, erweiterte Funktionalitäten dargestellt. Aus diesem Grund soll auch für die Desktop-Version der rechte Mausklick zum Anzeigen und Verstecken der zwei Buttons auf einem Listeneintrag verwendet werden.

III.2 Implementierung

III.2.1 Entwicklungsumgebung

Für die prototypische Umsetzung der Taskmanager-App wird auf das von Adobe bereitgestellte Flex SDK und den Adobe Flash Builder zurückgegriffen. Beide Werkzeuge wurden grundlegend für die Entwicklung mobiler AIR Applikationen optimiert. Da der Prototyp konzeptionell auf ein View-basiertes Modell setzt, kann insbesondere mit den neuen Mobil-Komponenten des Flex SDK gearbeitet werden. Dies soll im Folgenden genauer ausgeführt werden.

Framework: Adobe® Flex® SDK 4.5.1

Auf den Core APIs des Flash Players und AIR aufbauend, stellt das Flex SDK ein robustes Framework zum Erstellen von Interfaces und serverseitgen Datenverbindungen zur Verfügung. Flex beinhaltet ein großes Set an integrierten UI-Komponenten, unterstützt deklarative UI-Erstellung mittels MXML, dynamische Layouts sowie eine erweiterbare Komponenten-Architektur. [Jaramillo01 2011] Mit der Version 4.5 des Flex SDKs wurde dieses für die mobile Entwicklung um neue Features erweitert. Aufgrund geringerer Ressourcen und kleinerer Screengrößen der mobilen Endgeräte wurden vorhandene Komponenten um mobile Skins und touch-basierte Interaktionen ergänzt. Buttons, Checkboxen oder Textinputfelder wurden so in ihrer Größe für Touch-Eingaben optimiert, Listen und scrollbare Bereiche durch Touch-and-Throw Scrolling an das Verhalten nativer Listen-Elemente angepasst [Jaramillo01 2011]. Zur Unterstützung der mobilen Screen Metapher wurde das Flex SDK durch die Komponenten View, ViewNavigator und TabbedViewNavigator erweitert.

Abbildung 4.1: Eine Auswahl der mobilen Komponenten in Flex 4.5 [Jaramillo 2011]

■ TOUCH AND THROW: Zurückschnellen des Inhalts am Ende des scrollbaren Bereiches.

Seit dem offiziellen Release des Flex SDK 4.5 im Mai 2011 unterstützte die API neben Desktop-Anwendungen auch das Erstellen von mobilen iOS und Android Applikationen. Dabei blieb die Performance auf iPhone und iPad weit hinter der auf Android-Geräten zurück. Außerdem konnte beim Entwickeln von iOS Apps nicht auf Flex zurückgegriffen werden, da die Komponenten zunächst nur für Android Anwendungen vorgesehen waren.

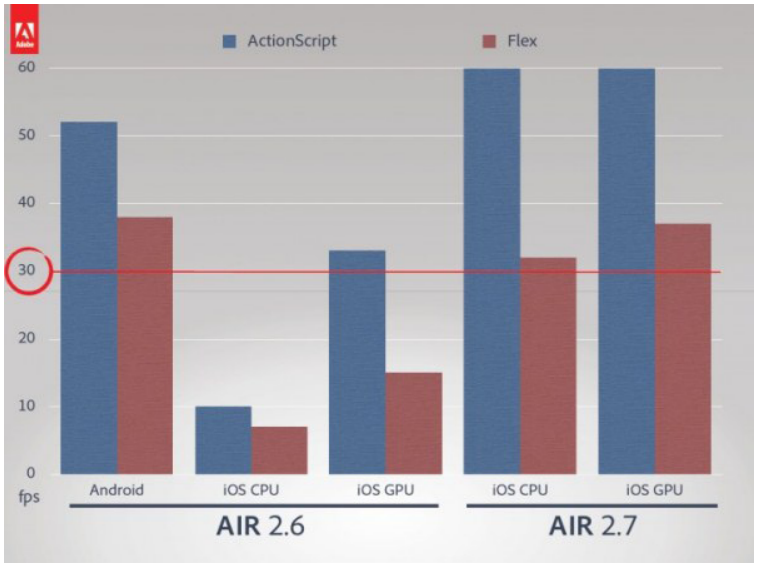


Abbildung 4.2: Performancevergleich zwischen AIR v2.6 und AIR v2.7 [Sonnati 2011]

Mit dem Update vom 20. Juni 2011 auf die aktuelle Version 4.5.1 verfügt das Flex SDK nun über einen aktualisierten iOS-Packager, welcher einen enormen Performancegewinn für Flex und ActionScript Applikationen auf iPhone, iPod und iPad verspricht (vgl. Abbildung 4.2). Weiterhin wurde durch die Integration eines RIM-Plugins das Erstellen von Blackberrys Playbook Apps ermöglicht. [Shorten01 2011] Der aktuelle Stand des Flex SDK 4.5.1 beinhaltet das AIR SDK 2.7, welches im selben Zuge akualisiert wurde. Für die vorliegende Arbeit wird diese Version als Untersuchungsstand festgelegt und künftige Updates nicht weiter betrachtet. Das AIR SDK unterstützt in dieser Version die Desktopsysteme Windows, Mac OS und Linux, sowie die Mobilsysteme iOS, Android und Blackberry Tablet OS.

Entwicklungsumgebung: Adobe® Flash® Builder 4.5™

Der Adobe Flash Builder ist eine auf Eclipse aufgesetzte Entwicklungsumgebung, welche sämtliche für die Entwicklung von Flex- und ActionScript

3.0 Applikationen benötigten Werkzeuge zur Verfügung stellt. Flash Builder läuft unter Microsoft Windows sowie Mac OS X und ist in verschiedenen Versionen verfügbar. Die IDE lässt sich optional als Eclipse-Plugin oder als Standalone Anwendung mit integriertem Eclipse Arbeitsbereich installieren. [Adobe05 2010]

Seit 3. Mai 2011 ist die neue Version 4.5 des Flash Builders, die zuvor unter dem Entwicklungsnamen „Burrito“ bekannt war, erhältlich. Wie das Flex SDK erhielt auch die zugehörige IDE ein umfangreiches Update. So werden Designern und Entwicklern nun zusätzlich zu Web und Desktop neue Tools und Workflows für die Entwicklung von Smartphone und Tablet Apps bereitgestellt [Subramaniam 2011]. Dabei unterstützt der Flash Builder seit Juni 2011 das Entwickeln, Testen und Veröffentlichen von mobilen Anwendungen für die Zielplattformen iOS (iPad, iPhone ab 3GS), Android (ab v2.2) und Blackberry Tablet OS. Über den vorhandenen Design-Mode kann das Verhalten des deklarativen Layouts in Flex für verschiedene Displayauflösungen und Device-Orientierungen vor dem Kompilieren getestet werden (vgl. Abbildung 4.3).

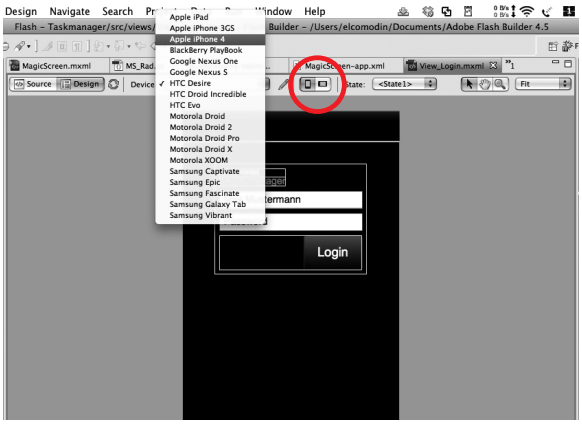


Abbildung 4.3: Flash Builder 4.5 Design Mode: Einstellmöglichkeiten für Geräte und Orientierung

■ DEKLARATIV: Beschreibung der Anzeige und Anordnung von Elementen mittels MXML.

■ DEVICE-ORIENTIERUNG: Mobilgeräte können hochkant oder quer verwendet werden.

Anlegen mobiler Projekte mit Adobe Flash Builder 4.5

Flash Builder stellt zwei Typen für mobile Projekte bereit: ActionScript Mobile Project und Flex Mobile Project (vgl. Abbildung 4.4). Für beide Typen sind defaultmäßig alle Zielplattformen ausgewählt und können manuell deaktiviert werden. Beim Verwenden eines mobilen Flex-Projektes erhält der Entwickler Zugang zum Flex Framework, inklusive der neuen mobilen Flex Komponenten und Navigation-Features sowie zum Desgin Mode und den intergrierten Data-Services [Jaramillo01 2011]. Mobile ActionScript Projekte dagegen liefern eine leere Main-Klasse, die Ausgangpunkt für ein reines ActionScript Projekt ohne Flex ist.

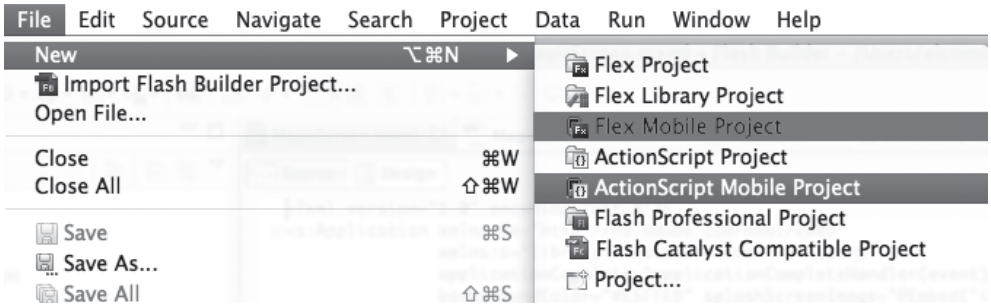


Abbildung 4.4: Flash Builder 4.5: Neue Projekttypen für Mobile Apps

Beim Erstellen eines mobilen Flex-Projektes gibt es weitere Einstellungsmöglichkeiten. Flash Builder stellt drei verschiedene Templates für das grundlegende Applikationslayout bereit: Blank, View-Based Application und Tabbed Application (vgl. Abbildung 4.5).

Weiterhin lassen sich hier für das Verhalten der App auf den jeweiligen Zielplattformen Voreinstellungen treffen. Dies kann beispielsweise die Zugriffsrechte betreffen, aber auch Screen-Auflösung, Vollbildmodus oder automatische Reorientierung des Screens können festgelegt werden. Über eine Konfigurations-XML können alle Einstellungen auch während der Entwicklung definiert werden. Details zu den Konfigurationsmöglichkeiten werden in den folgenden Kapitel beschrieben.

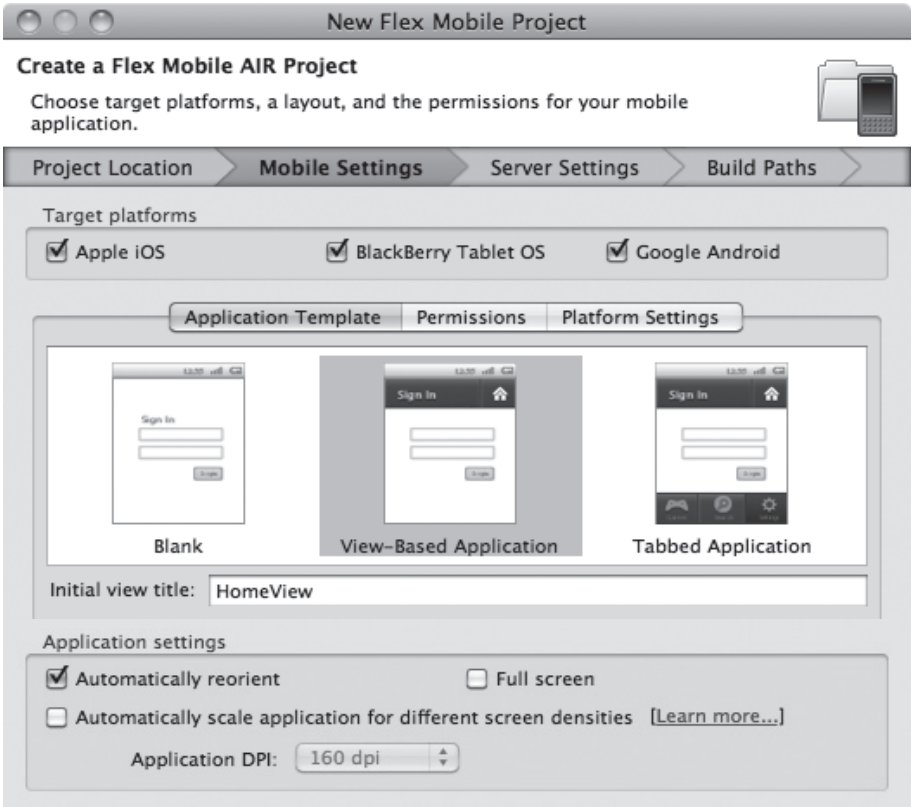


Abbildung 4.5: Flash Builder 4.5: Auswahl der Zielplattformen und Konfigurationen

Erstellen mobiler Projekte mit Adobe® AIR® LaunchPad

Für einen schnellen Einstieg in die Erstellung mobiler AIR Projekte mit Flash Builder veröffentlichte Adobe das Helfer-Programm „Adobe AIR Launchpad“[9]. Mit dem Launchpad „lassen sich Projekte wie im Flash Builder, aber mit deutlich mehr Einstellungsmöglichkeiten anlegen. [So] können [] über einfache Dialoge nicht nur die Permissions (wie Zugriff aufs Internet oder GPS-Daten) definiert [werden], sondern auch Samples aktiviert, die beim Anlegen des Projekts Vorlagen für verschiedene Funktionen im Code platzieren.“ [Widjaja01 2011] Dies sind beispielsweise Code-Snippets zum Einbinden der Kamera, GPS oder Multitouch-Gesten (vgl. Abbildung 4.6). Abschließend erzeugt Adobe Launchpad ein fertiges Flash Builder Projekt, welches in die IDE importiert und sofort kompiliert werden kann.

■ CODE-SNIPPET: Ein kurzes Stück Quellcode.

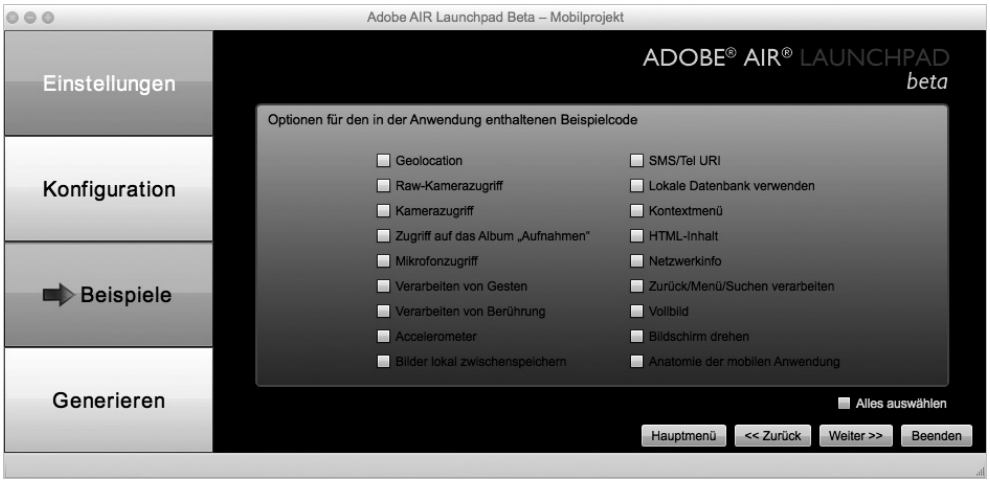


Abbildung 4.6: Adobe AIR Launchpad: Konfigurationsmöglichkeiten für Beispiel-Code

Testen mobiler Projekte

Zum Testen und Debuggen eines Projekts bietet der Flash Builder zwei Optionen. Zum einen kann die App in einem Emulator auf dem Desktop unter Verwendung des in die IDE integrierten AIR Debug Launcher (ADL) ausgeführt werden. Zum anderen kann die Ausführung der Applikation direkt auf dem physikalischen Gerät erfolgen. Beide Varianten ermöglichen Zugriff auf den vollen Funktionsumfang der Flash Builder Debug-Funktionalitäten, wie zum Beispiel das Setzen von Breakpoints. [Jaramillo01 2011]

9 http://labs.adobe.com/technologies/airlaunchpad/

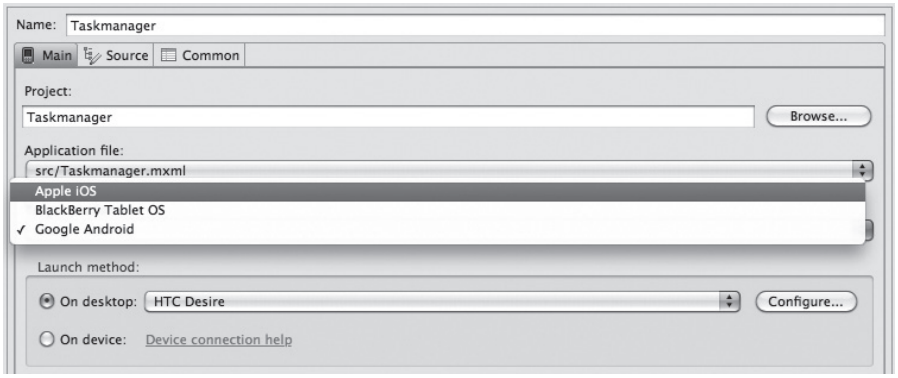


Abbildung 4.7: Flash Builder 4.5: Run Konfigurationen

Für schnelles Testen ist das Ausführen im Emulator vorzuziehen. Dies ist jedoch nur möglich, solange keine Device-spezifischen Schnittstellen, wie Beschleunigungssensor oder GPS, angesprochen werden, da diese kaum oder gar nicht auf dem Desktop simuliert werden können. Flash Builder stellt für den Emulator bereits eine Reihe von Geräte-Konfigurationen für verschiedene Zielplattformen bereit (vgl. Abbildung 4.8). Sie können problemlos um eigene Definitionen erweitert werden.

Device Configurations

Add, remove, and edit device configurations.

Device Name	Platform	Full Screen Size	Usable Portrait Size	Usable Landscape Siz	Pixels Per Inch (PPI)
Apple iPad	Apple iOS	768 x 1024	768 x 1004	1024 x 748	132
Apple iPhone 3GS	Apple iOS	320 x 480	320 x 460	480 x 300	163
Apple iPhone 4	Apple iOS	640 x 960	640 x 920	960 x 600	326
BlackBerry PlayBook	BlackBerry Tablet OS	1024 x 600	600 x 1024	1024 x 600	167
Google Nexus One	Google Android	480 x 800	480 x 762	800 x 442	252
Google Nexus S	Google Android	480 x 800	480 x 762	800 x 442	235
HTC Desire	Google Android	480 x 800	480 x 762	800 x 442	252
HTC Droid Incredible	Google Android	480 x 800	480 x 762	800 x 442	252
HTC Evo	Google Android	480 x 800	480 x 762	800 x 442	217
Motorola Droid	Google Android	480 x 854	480 x 816	854 x 442	265
Motorola Droid 2	Google Android	480 x 854	480 x 816	854 x 442	265
Motorola Droid Pro	Google Android	320 x 480	320 x 455	480 x 295	144
Motorola Droid X	Google Android	480 x 854	480 x 816	854 x 442	228
Motorola XOOM	Google Android	1280 x 800	800 x 1232	1280 x 752	150
Samsung Captivate	Google Android	480 x 800	480 x 762	800 x 442	233
Samsung Epic	Google Android	480 x 800	480 x 762	800 x 442	233
Samsung Fascinate	Google Android	480 x 800	480 x 762	800 x 442	233
Samsung Galaxy Tab	Google Android	600 x 1024	600 x 986	1024 x 562	168
Samsung Vibrant	Google Android	480 x 800	480 x 762	800 x 442	233

Abbildung 4.8: Flash Builder 4.5: Vorinstallierte Device Konfigurationen

Zusätzlich zum Emulator besteht die Möglichkeit, eine App live auf einem über USB verbundenem Gerät zu testen. Dabei sind für die verfügbaren Zielplattformen sehr unterschiedliche und teilweise aufwendige Voraussetzungen erforderlich. Kapitel III.3 wird sich den Details dieses Prozesses widmen.

III.2.2 Schwerpunkte

Durch die prototypische Umsetzung der Taskmanager App als Vertreter einer mobilen Adobe AIR Applikation sollen die im Theorieteil abgeleiteten Erwartungen überprüft werden. Vordergründiges Ziel dabei ist es, die Applikation für ihren jeweiligen Device-Context zu sensibilisieren und dadurch dynamisch und anpassungsfähig zu halten.

■ DEVICE-CONTEXT: Rahmenfaktoren der aktuelle Geräteumgebung einer App, wie z. B. Displayauflösung und Betriebssystem.

Für dieses sogenannte Kontext-Bewusstsein werden wesentliche Kriterien herangezogen und untersucht. Dies betrifft zum einen die Möglichkeit einer vorteilhaften Strukturierung des Projektes für mehrere Zielplattformen sowie mögliche Anbindungen benötigter nativer Gerätefunktionen. Zum anderen wird untersucht, welche Unterstützungsmaßnahmen durch das Framework zur Integration der mobilen Screen Metapher bereitgestellt werden. Ein dritter Schwerpunkt wird sich mit der Herausforderung beschäftigen, die App aus nur einer Codebasis heraus dynamisch an eine Vielzahl verschiedener Displayauflösungen anzupassen. Eine abschließende Prüfung soll feststellen, ob sich die App für alle in der statistischen Auswertung aufgestellten Zielplattformen exportieren lässt und welche Schritte dafür erforderlich sind.

III.2.2.1 Projektstruktur und -konfiguration

Vor dem Start der eigentlichen Programmierung einer mobilen AIR App ist es notwendig, eine sinnvolle Projektaufteilung- und konfiguration zu wählen. Dazu zählen unter anderem die Auslagerung gemeinsamer Funktionsbibliotheken, das Festlegen von Zugriffsberechtigungen sowie grundlegende Einstellungen bezüglich des Verhaltens der App auf dem Endgerät. Flash Builder 4.5 bietet dem Entwickler hierfür zahlreiche Unterstützungen, die im Folgenden vorgestellt werden.

Verwaltung von Multiscreen-Projekten

Für die Multiscreen-Entwicklung von Applikationen ist es praktikabel, den Code in verschiedenen Projekten zu verwalten. Ein normales Flex- oder ActionScript-Projekt enthält das User Interface für Desktop und/oder Web,

ein weiteres Flex- oder ActionScript-Projekt enthält das mobile UI und ein Library-Projekt enthält die Logik für Modell und Datenzugänge. Die verschiedenen UI-Projekte teilen sich über das Library-Projekt dieselben Funktionsbibliotheken. Somit wird redundanter Programmieraufwand erheblich minimiert. [Jaramillo01 2011]

■ LIBRARY PROJEKT: Dient der Verwaltung von Funktionsbibliotheken.

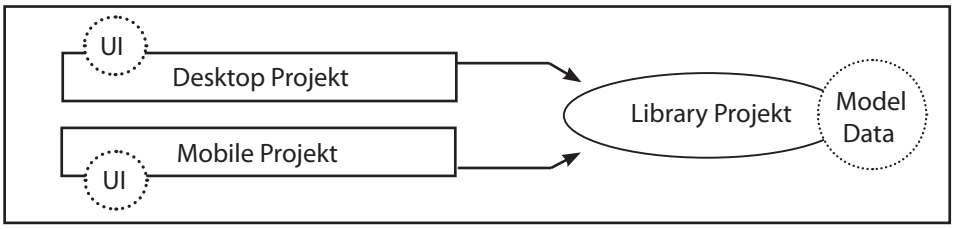


Abbildung 5.1: Typische Projektstruktur für eine Multiscreen-App

Dies ist ein eleganter Weg, sobald sich Grafik- und Usability-Konzept der Desktop- und Mobil-Variante stark unterscheiden. Die Taskmanager-App soll jedoch als Desktop-Widget über dasselbe UI verfügen wie ihre mobile Variante und lediglich wenige Anpassungen in der User-Interaktion erfordern. Ein separates UI-Projekt würde hier für viel doppelten Code sorgen, daher kommen für die Umsetzung des Taskmanagers nur zwei Projekte zum Einsatz: ein mobiles Flex-Projekt für das User-Interface und ein Library-Projekt zur Kapselung der Datenanbindungen. Das Projekt wird so für weitere UI-Varianten modular gehalten (vgl. Abbildung 5.2).

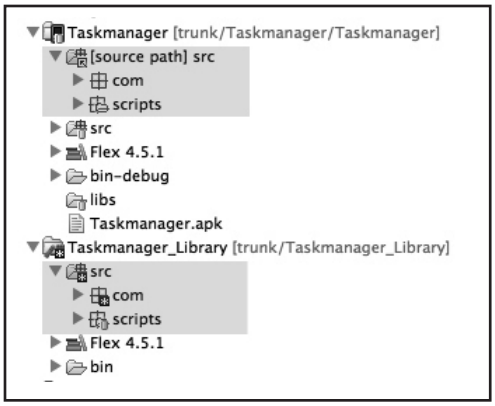


Abbildung 5.2: Taskmanager-Projekt und inkludiertes Taskmanager_Library-Projekt

Flash Builder 4.5 stellt für die Verwaltung von Daten und Funktionsmodulen einen eigenen Projekttyp zur Verfügung. Dieser kann über den Befehl File->New->Flex Library Project erstellt werden. Beim Konfigurieren kann zusätzlich unterschieden werden, ob das Projekt als generische Bibliothek für Web, Desktop und Mobilgerät fungieren soll oder ausschließlich für mobile Applikationen. Im mobilen Flex-Projekt kann das erstellte Library-Projekt anschließend über die Rubrik „Flex Build Path“ des Eigenschaften-Dialoges eingebunden werden (vgl. Abbildung 5.3).

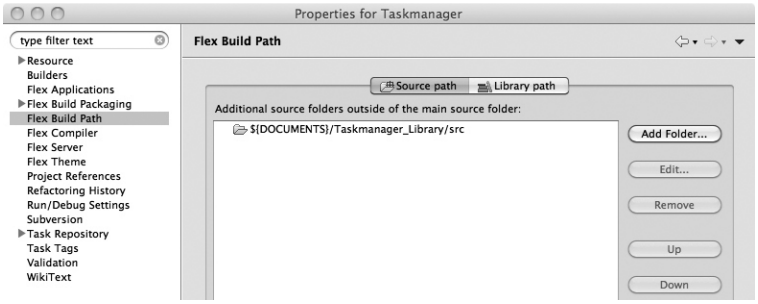


Abbildung 5.3: Flash Builder 4.5: Library-Projekt wird als Quelle für Taskmanager-Projekt hinzugefügt

Da sowohl die mobile Taskmanager-App als auch das Taskmanager Desktop-Widget aus derselben Codebasis generiert werden sollen, ist es notwendig zur Laufzeit unterscheiden zu können, auf welchem Gerätetyp die Anwendung ausgeführt wird. Die AIR-API stellt zu diesem Zweck eine einfache Abfrage bereit.

```
if(Capabilities.touchscreenType == TouchscreenType.NONE)
  Multitouch.inputMode = MultitouchInputMode.NONE;

if(Capabilities.touchscreenType == TouchscreenType.STYLUS)
  Multitouch.inputMode = MultitouchInputMode.GESTURE;

if(Capabilities.touchscreenType == TouchscreenType.FINGER)
  Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
```

Listing 2.1: Taskmanager.mxml, TouchscreenType

Wird der Code auf einem nicht multitouchfähigen Desktop ausgeführt, wird `Capabilities.touchscreenType` auf `NONE` gesetzt. An dieser Stelle können dann im Code Fallunterscheidungen implementiert werden und statt `TouchEvents` beispielsweise `MouseEvents` aktiviert werden.

Wird ein ursprünglich für den Desktop konzipiertes AIR-Projekt auf einem Multitouch-Gerät ausgeführt, werden einfache Single-Finger Taps automatisch durch die AIR-API auf entsprechende MouseEvents abgebildet [Goldmann 2011]. Für simple Anwendungen, die ohne komplexe Multitoch-Gesten auskommen, muss demzufolge nicht zwangsläufig eine Fallunterscheidung erfolgen.

Aufbau eines mobilen Flex-Projektes

Wie bereits im Kapitel III.2.1 erwähnt, stellt Flash Builder 4.5 verschiedene Typen mobiler Flex-Projekte bereit. Aufgrund des View-basierten Konzepts des Taskmanagers wird die App mit dem Typ „View-Based Application“ an-

gelegt. Auf die Besonderheiten dieses Projekttyps sowie die bestehender Alternativen wird im nächsten Kapitel genauer eingegangen.

Ein mobiles Flex-Projekt vom Typ „View-Based“ besteht zunächst aus vier Files und zwei Paketen. Im „default package“ befindet sich die initiale MXML-Datei. Sie ist vom Typ `<s:ViewNavigatorApplication/>` und hält ein einfaches Framework zur Integration des View-basierten Navigationsmodels bereit [Adobe06 2011]. Diese Klasse bildet den Haupteinstiegspunkt in die Applikation und steuert das Laden, Anzeigen und Überblenden der individuellen Views. Über das Attribut `firstView` wird die erste darzustellende View festgelegt.

```
<?xml version="1.0" encoding="utf-8"?>

<s:ViewNavigatorApplication
   xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   firstView="views.View_Login">

</s:ViewNavigatorApplication>
```

Listing 2.2: Taskmanager.mxml, Root-Tag

■ TAG: Bezeichnung eines XML-Elementes.

Diese sogenannte „HomeView“ ist nach dem Anlegen des Projektes bereits im Paketordner „views“ vorhanden und kann direkt weiterbearbeitet werden. Dadurch ist das Projekt von Beginn an kompilierfähig und kann sofort im Emulator oder auf einem externen Gerät getestet werden. Die XML-Files „blackberry-tablet.xml“ und „Taskmanager-app.xml“ dienen dem Einstellen von allgemeinen gerätebezogenen Konfigurationen sowie der Zugriffsberechtigungen. Die Konfigurations-XML (auch Deskriptor-XML) „Taskmanager-app.xml“ ist die Hauptinformationsquelle der Anwendung. Hier können Beschreibungen hinzugefügt und bestimmte Verhaltensweisen der Applikation definiert werden.

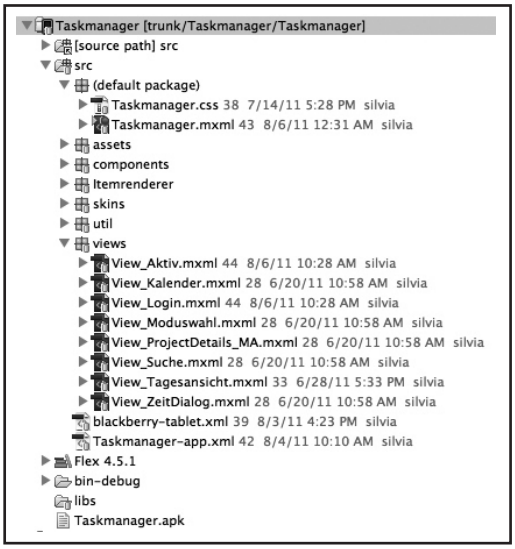


Abbildung 5.4: Paketstruktur des Taskmanager Flex Mobile Projektes

Zugriffsberechtigungen

Beim Publizieren mobiler Apps ist es sinnvoll, diesen nur so viel Zugriffsrechte auf das System einzurichten wie nötig. Der User wird vor dem Installieren der App automatisch vom System über alle zum Ausführen der App benötigten Zugriffsarten informiert und muss seine Kenntnisnahme bestätigen.

Für die drei durch Flash Builder 4.5 aktuell unterstützten Systeme werden verschiedene Möglichkeiten zur Konfiguration dieser Berechtigungen vorgegeben. Für iPhone und iPad beispielsweise können keinerlei separate Einstellungen vorgenommen werden. An der entsprechenden Stelle im Projekteigenschaften-Editor wird der Entwickler darauf hingewiesen, dass die Applikation zur Laufzeit automatisch benötigte Systemzugriffe erkennt und den Anwender um Erlaubnis fragen wird.

Für Android und Blackberry Tablet OS ist dies nicht der Fall. Hier müssen alle Systemzugriffe über spezifische Einstellungsdialoge explizit vom Entwickler definiert werden. Für Android Applikationen können die Zugriffsarten an verschiedenen Stellen im Projekt festgelegt werden. Dies kann entweder über einen grafischen Dialog direkt beim Erstellen eines neuen Projektes (vgl. Abbildung 5.5) erfolgen oder innerhalb des Android-Blocks `<android>` der bereits erwähnten Konfigurations-XML „Taskmanager-app.xml“ vorgenommen werden (vgl. Listing 2.3). Bisher sind für Android folgende Berechtigungen einstellbar:

- INTERNET
  Allows applications to open sockets and embed HTML content.
- WRITE_EXTERNAL_STORAGE
  Allows an application to write to external storage.
- READ_PHONE_STATE
  Allows the AIR Runtime to mute audio from application, in case of incoming call.
- ACCESS_FINE_LOCATION
  Allows an application to access GPS location.
- DISABLE_KEYGUARD, WAKE_LOCK
  Allows applications to access screen dimming provision.
- CAMERA
  Allows applications to access device camera.
- RECORD_AUDIO
  Allows applications to access device microphone.
- ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE
  Allows applications to access information about network interfaces associated with the device.
  [Tretola 2011]

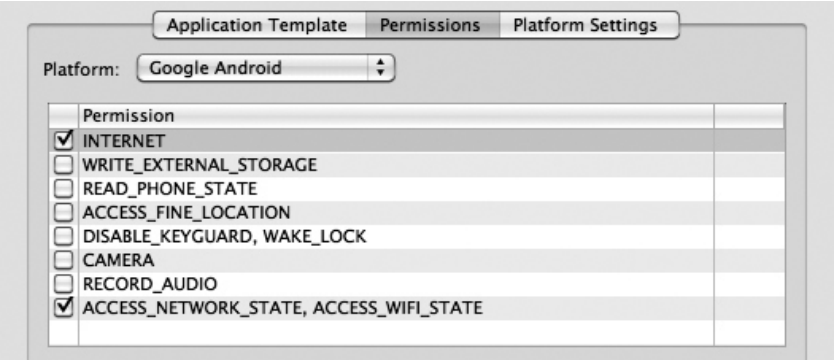


Abbildung 5.5: Flash Builder 4.5: Zugriffseinstellungen für Android

```
<android>
  <manifestAdditions><![CDATA[
  <uses-permission android:name=
         "android.permission.INTERNET"/>
  <!--<uses-permission android:name=
         "android.permission.WRITE_EXTERNAL_STORAGE"/>-->
  <!--<uses-permission android:name=
         "android.permission.READ_PHONE_STATE"/>-->
  <!--<uses-permission android:name=
         "android.permission.ACCESS_FINE_LOCATION"/>-->
  <!--<uses-permission android:name=
         "android.permission.DISABLE_KEYGUARD"/>-->
  <!--<uses-permission android:name=
         "android.permission.WAKE_LOCK"/>-->
  <!--<uses-permission android:name=
         "android.permission.CAMERA"/>-->
  <!--<uses-permission android:name=
         "android.permission.RECORD_AUDIO"/>-->
  <uses-permission android:name=
         "android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name=
         "android.permission.ACCESS_WIFI_STATE"/>
  </manifest>
  ]]></manifestAdditions>
</android>
```

Listing 2.3:Taskmanager-app.xml, Android-Einstellungen

Einzige Voraussetzungen für die Taskmanager-App sind Internetzugang und Kenntnis über den Netzwerkstatus.

Für das Blackberry Tablet OS sehen die Einstellungsmöglichkeiten ähnlich aus. Sie sind über den Erstellungsdialog konfigurierbar, welcher auch im Entwicklungsverlauf über die Projekteigenschaften aufrufbar ist (vgl. Abbildung 5.6). Definierte Zugriffsarten werden durch die IDE automatisch in die Blackberry Konfigurationsdatei „blackberry-tablet.xml" eingetragen (vgl. Listing 2.5)

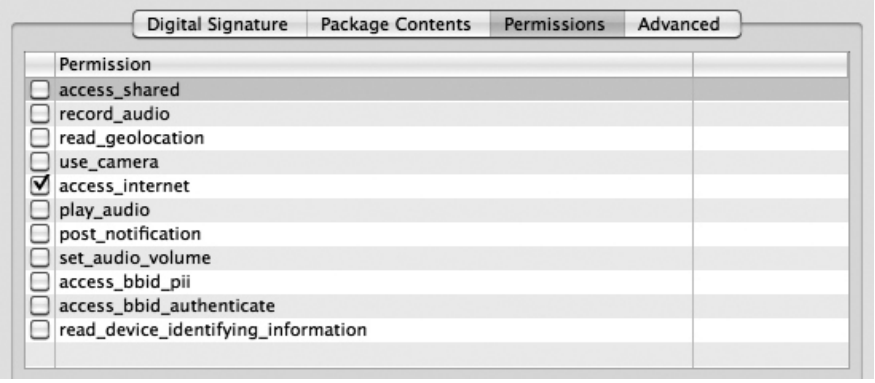


Abbildung 5.6: Flash Builder 4.5: Zugriffseinstellungen für Blackberry Tablet OS

Weitere Konfigurationsmöglichkeiten

Neben üblichen Eigenschaften wie Breite, Höhe, Titel und Version können über die Deskriptor-XML auch spezifische Einstellungen bezüglich des Verhaltens der App auf dem Endgerät vorgenommen werden. Dies betrifft beispielsweise die Ausrichtung der grafischen Komponenten entsprechend der aktuellen Orientierung des Gerätes. Für den Taskmanager wurden folgende Konfigurationen festgelegt:

```
<aspectRatio>portrait</aspectRatio>
<autoOrients>true</autoOrients>
<fullScreen>true</fullScreen>
<renderMode>cpu</renderMode>
<softKeyboardBehavior>pan</softKeyboardBehavior>
```

Listing 2.4: Taskmanager-app.xml, Einstellungsmöglichkeiten

Die Eigenschaft `aspectRatio` bestimmt die standardmäßge Ausrichtung der App beim Programmstart. Dies kann entweder `portrait` oder `landscape` sein. Über die Eigenschaft `autoOrients` kann zusätzlich festgelegt werden, ob sich die Anwendung automatisch an die Ausrichtung des Gerätes anpasst. Ist `autoOrients` auf `true` gesetzt und das Handy oder Tablet wird vom User gedreht, sendet die App ein entsprechendes Event und der Entwickler kann das Format der View anpassen. Das Attribut `fullScreen` gibt an, ob die App im Vollbildformat gestartet wird oder die systemeigene Statusleiste weiterhin verfügbar bleibt. Der nutzbare Bereich ist dann um die Maße der Statusleiste eingeschränkt. Leider ist die Umsetzung dieses Attributs zwischen den Zielplattformen nicht einheitlich. Unter Android 2.x und iOS wird tatsächlich über der App eine Statusbar angezeigt, wenn `fullScreen==false` und im Vollbildmodus aus-

- PORTRAIT: Hochformat.
- LANDSCAPE: Querformat.

geblendet. Unter Android 3.x ist die Statusbar am unteren Bildschirmrand jedoch immer sichtbar, unabhängig ob Vollbild aktiviert wurde oder nicht. Applikationen auf dem Blackberry Tablet wiederum laufen standardmäßig im Vollbildmodus. [Jose01 2011]

Mit der Eigenschaft `renderMode` kann festgelegt werden, ob beim Rendern zusätzlich der Hardwarebeschleuniger des Gerätes verwendet werden soll, um eine bessere Performance zu erreichen. Hierzu wird `gpu` als Wert eingetragen. Dies ist besonders dann zu empfehlen, wenn die App eine Vielzahl von Bitmap-Grafiken enthält. CPU-Rendering ist dagegen der traditionelle Ansatz zum Rendern von vektorbasierten Inhalten und kommt daher auch für den Taskmanager zum Einsatz.

■ GPU: „Graphics Processing Unit" - Grafikprozessor.

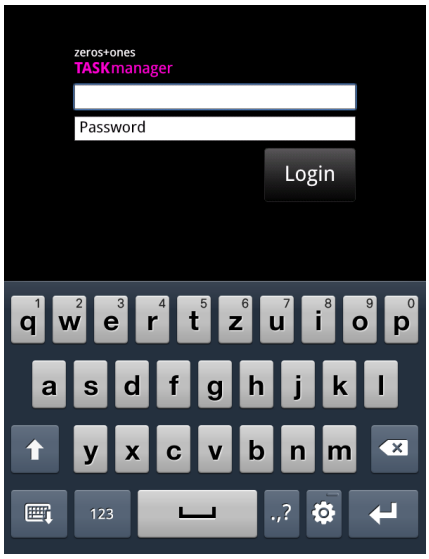


Abbildung 5.7: Taskmanager-App unter Android mit aktiviertem soft keyboard

Mobile Flex-Projekte sind von Beginn an so konfiguriert, dass die Fokussierung eines Textfeldes automatisch detektiert und daraufhin das onscreen keyboard (techn. soft keyboard) des jeweiligen Systems aktiviert wird. Da die Tastatur einen beträchtlichen Teil des Displays einnimmt, ist es unvermeidbar, dass einige Elemente von ihr überdeckt werden. Um diesem Verhalten entgegenzuwirken, verschiebt AIR die Anwendungsbühne so, dass sowohl Textfeld als auch soft keyboard sichtbar bleiben. Eine eigene Reaktion kann implementiert werden, indem die Eigenschaft `softKeyboardBehavior` der Konfigurations-XML auf `none` gesetzt wird. [Jose01 2011]

Die AIR-API stellt für die Verwendung des onscreen keyboards weitere Funktionen zur Verfügung. So kann dieses über den Befehl `InteractiveObject.requestSoftKeyboard()` für beliebige `InteractiveObjects` angefordert werden, ohne dass diese ursprünglich der Textverarbeitung dienten. Allerdings ist diese Option nicht in allen Systemen verfügbar:

■ INTERACTIVE OBJECT: Abtrakte Basisklasse aller interaktiven Display-Objekte in ActionScript 3.

> „Note: This method is not supported in AIR applications on iOS". [Adobe06 2011]

Gerätespezifische Einstellungen für das Blackberry Tablet OS werden in der separaten Datei „blackberry-tablet.xml“ vorgenommen. Eigenschaften wie `<author>` oder `<image>` sind für die spätere Veröffentlichung der Anwendung in der Blackberry App World obligatorisch.

```
<?xml version="1.0" encoding="UTF-8"?>
<qnx>
  <icon>
     <image>img/appIcon.png</image>
   </icon>
   <author>Silvia Graumann</author>
   <category>core.games</category>
   <splashscreen>img/splash.png</splashscreen>
   <buildId>349</buildId>
   <platformVersion>1.0.0.1</platformVersion>
   <permission>access_internet</permission>
</qnx>
```

Listing 2.5: blackberry-tablet.xml

Unterstützte Systemschnittstellen

Das neue AIR SDK 2.7 unterstützt den Zugang zu vielen gerätespezifischen Funktionen. Einige davon, wie Beschleunigungssensor, Screen Ausrichtung und Multitouch Support ergänzen erst seit kurzer Zeit die AIR-API und wurden bereits im Kapitel über die Grundlagen mobiler AIR-Projekte angesprochen. Andere, eher grundlegende Informationen, die zur Laufzeit von großem Nutzen sein können, bündelt AIR bereits seit Version 1.0 in der Klasse `flash.system.Capabilities`. Für die plattformübergreifende Entwicklung mobiler Applikationen erlangt diese Klasse eine neue Bedeutung. So kann eine App unterschiedliche Darstellungsoptionen beinhalten, je nach dem, auf welchem Betriebssystem diese gerade ausgeführt wird, wie hoch die Screen-Auflösung oder wer der Gerätehersteller ist. Folgende Eigenschafen geben darüber Auskunft:

```
Capabilities.os
Capabilities.screenDPI
Capabilities.manufacturer
```

Listing 2.6: flash.system.Capabilities, ausgewählte Eigenschaften

AIR bietet weiterhin Schnittstellen zur Ermittlung der User-Aktivität sowie anderer Systemänderungen, wie die Abweichung der GPS-Position und viele weitere (vgl. Tabelle 1.2).

Systemschnitt-stellen unter AIR	Multitouch	Netzwerk	GPS	Accelerometer	Camera	Image Gallery	Video Recording	Bluetooth	Local Storage	SdCard	Microphone	Sound	Sound Recording	Vibration Alert
verfügbar	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	

Systemschnitt-stellen unter AIR	Hardware Buttons	Kalenderzugriff	Adressbuch (write)	SMS	Email	Phone Calls	Kompass	File System	Barcode Reader	Native Maps	Built-In HTML	Detect Exiting	Screen Orientation
verfügbar	✓			✓	✓	✓		✓	(✓)	(✓)	✓	✓	✓

■ (✓): Möglich durch externe Bibliotheken

Tabelle 1.2: Unterstützung von Systemschnittstellen durch die AIR-API

Auch die Unterstützung von Multitasking beziehungsweise die Erkennung des Aktivierens und Deaktivierens einer AIR App wurde durch Adobe in die neue SDK Version aufgenommen. Diese Funktionalität ist für den Prototyp von großer Bedeutung. So ist eine der wichtigsten Funktionen der Taskmanager-App die automatische Zeiterfassung. Sobald für ein bestimmtes Projekt der „Start"-Button gedrückt wurde, wird ein Timer gestartet und die seit dem Start vergangenen Minuten werden gezählt. Da ein Mitarbeiter an einem Projekt auch über mehrere Stunden arbeiten kann, ist es nicht unwahrscheinlich, dass während dieser Zeit die Taskmanager-App durch Starten einer anderen Applikation oder das automatische Ausschalten des Displays deaktiviert wird. Dies ist ein gängiges Verhalten auf Mobilgeräten. Wichtig ist in diesem Fall lediglich, dass der Timer dabei nicht gestoppt wird, also die App im Hintergrund weiter laufen kann.

> Werden AIR-Anwendungen durch User-Interaktion oder durch das System in den Hintergrund verwiesen, werden sie durch das automatische Senden eines DEACTIVATE Events darüber informiert. Dementsprechend wird beim Aktivieren einer App ein ACTIVATE Event abgesendet. Das Verhalten

> der Anwendung in beiden Fällen variiert je nach Plattform. Wird eine AIR App unter Android deaktiviert, wird deren Animations-Framerate auf vier Frames pro Sekunde reduziert und die Renderingphase ausgelassen. Sämtliche Ereignisse werden jedoch weiterhin gesendet, wodurch Hintergrund-Prozesse wie das Beenden eines Uploads oder periodische Synchronisationen möglich bleiben. Unter iOS funktioniert das Multitasking-Modell etwas anders. Applikationen können normalerweise nicht im Hintergrund laufen, es sei denn sie performen einen bestimmten Hintergrund-Prozess-Typ wie beispielsweise das Aufrechterhalten eines voice-over-IP Gesprächs. Diese Art von Multitasking-Modell wird jedoch durch AIR momentan nicht unterstützt, so dass deaktivierte AIR Apps auf dem iPhone oder iPad komplett angehalten werden, ihre Framerate auf null geht und keinerlei Events mehr gesendet werden. Einzig der letzte Screen bleibt im Arbeitsspeicher erhalten, so dass bei erneuter Aktivierung der letzte Zustand der App wieder hergestellt werden kann. [Goldman 2011]

■ VOICE-OVER-IP: Führen von Telefonaten über das Internet.

Für das Verhalten des Taskmanagers bedeutet dies folgendes: Wie in Abbildung 5.8 zu sehen, wurde unter Android die Zeit für ein Telefonat gestartet. Bei Sekunde 00:07 wurde der Taskmanager durch Starten der Google Maps App in den Hintergrund gedrängt. Beim erneuten Aktivieren des Taskmanagers ist zu sehen, dass die Zeit für das Telefonat weiterlief, der Zeitstempel auf dem Telefon-Button gibt nun 02:11 an. Dieselbe Funktion auf einem iPad2 getestet zeigt ein anderes Verhalten. Nach dem Aktivieren der Taskmanager-App zeigt der Button immer noch 00:07 an und die Zeit wird nun von hier weiter gezählt. Lösbar ist dieses Problem momentan nur über einen Work-Around. Die Systemzeit muss beim Deaktivieren der App gespeichert und die bis zum erneuten Aktivieren vergangene Zeit auf den Timer addiert werden. Das Verhalten unter Blackberry Tablet OS konnte mangels eines realen Testgerätes leider nicht überprüft werden.

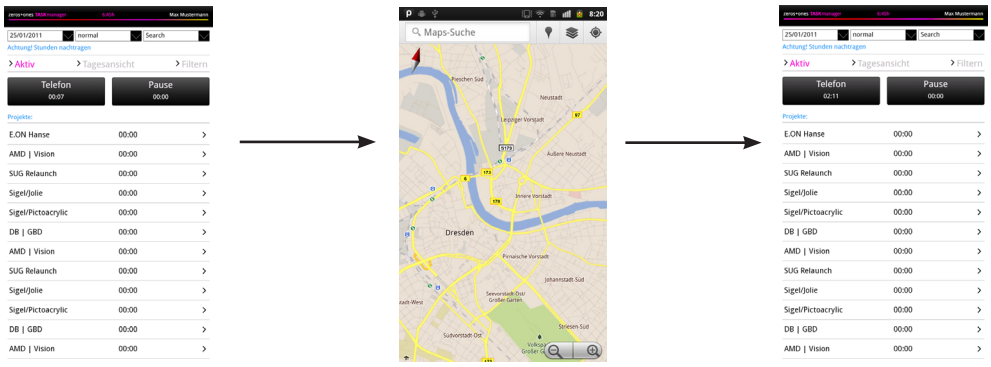


Abbildung 5.8: Deaktivieren der Taskmanager-App durch Starten einer anderen App unter Android

III.2.2.2 View-Navigator Prinzip

Zur Unterstützung der mobilen Screen Metapher hält das aktuelle Flex SDK neue Komponenten bereit. Wie bereits grundlegend erwähnt, beschreibt die Screen Metapher auf Mobilgeräten ein view-basiertes Navigationsmodell, welches durch die User-Interaktion zwischen einer Reihe von Vollbild-Views charakterisiert ist. Dieses Paradigma wird durch die meisten mobilen Applikationen integriert und innerhalb der einzelnen nativen Programmiersprachen mit Hilfe eines eingebetteten View-Navigation-Containers umgesetzt. [Adobe06 2011] Auch das Flex Framework unterstützt dieses Modell seit diesem Jahr und bietet hier einen eindeutigen Vorteil gegenüber ActionScript-Projekten, bei welchen die komplette View-Navigation manuell implementiert werden müsste.

View-Navigator Templates

■ TEMPLATE: Layout-Vorlage.

Für die Integration des View-Navigator Prinzips in die eigene Anwendung stehen drei Layout-Vorlagen zur Verfügung. Diese sind Blank Application, View-Based Application und Tabbed Application (vgl. Abbildung 5.5). Abhängig vom gewählten Template werden durch Flash Builder 4.5 automatisch verschiedene Files angelegt. Das Template „Blank Application“ enthält keine Unterstützung für eine View-Navigation und eignet sich daher am besten, um ein eigenes Navigationsmodell zu implementieren. Es wird lediglich die Haupteinstiegsklasse vom Typ <s:Application> generiert. [Tretola 2011]

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  applicationDPI="160">
</s:Application>
```

Listing 3.1: BlankApplication.mxml

Das View-Based Application Template basiert auf der neuen Komponente `<s:ViewNavigator>`. Das Konzept eines ViewNavigators besteht darin, eine Menge von einzelnen View Objekten zu verwalten, darzustellen und die Navigation zwischen den Views zu kontrollieren [Adobe06 2011]. Dabei

werden die Views intern durch einen Stack repräsentiert, wobei jeweils nur die oberste View des Stapels sichtbar ist.

■ STACK: Stapelverarbeitung von Objekten.

Abbildung 6.1: Tabbed Application, Flash Builder 4.5: Designansicht

Das dritte Template ist die Tabbed Application, welche durch die Komponente `<s:TabbedViewNavigatorApplication/>` umgesetzt wird. Dieses Template unterstützt die Unterteilung der App-UI in mehrere Tabs zur besseren Strukturierung besonders umfangreicher Anwendungen mit vielen Screens. Jeder Tab integriert wiederum einen eigenen ViewNavigator und kann darüber auch eine Startview festlegen. [Corlan 2011]

```
<s:TabbedViewNavigatorApplication
xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s=
"library://ns.adobe.com/flex/spark">

  <s:ViewNavigator label="FirstTab"
   firstView="views.FirstTabView"/>

  <s:ViewNavigator label="SecondTab"
   firstView="views.SecondTabView"/>

  <s:ViewNavigator label="ThirdTab"
   firstView="views.ThirdTabView"/>

</s:TabbedViewNavigatorApplication>
```

Listing 3.2: TabbedApplication.mxml

Da die Taskmanager-App keine Tabnavigation benötigt, wurde hier das View-Based Application Template eingesetzt. Abbildung 6.2 zeigt eine schematische Übersicht der Zusammenhänge sowie der verwendeten Navigationsrichtungen zwischen den einzelnen Views des Taskmanagers.

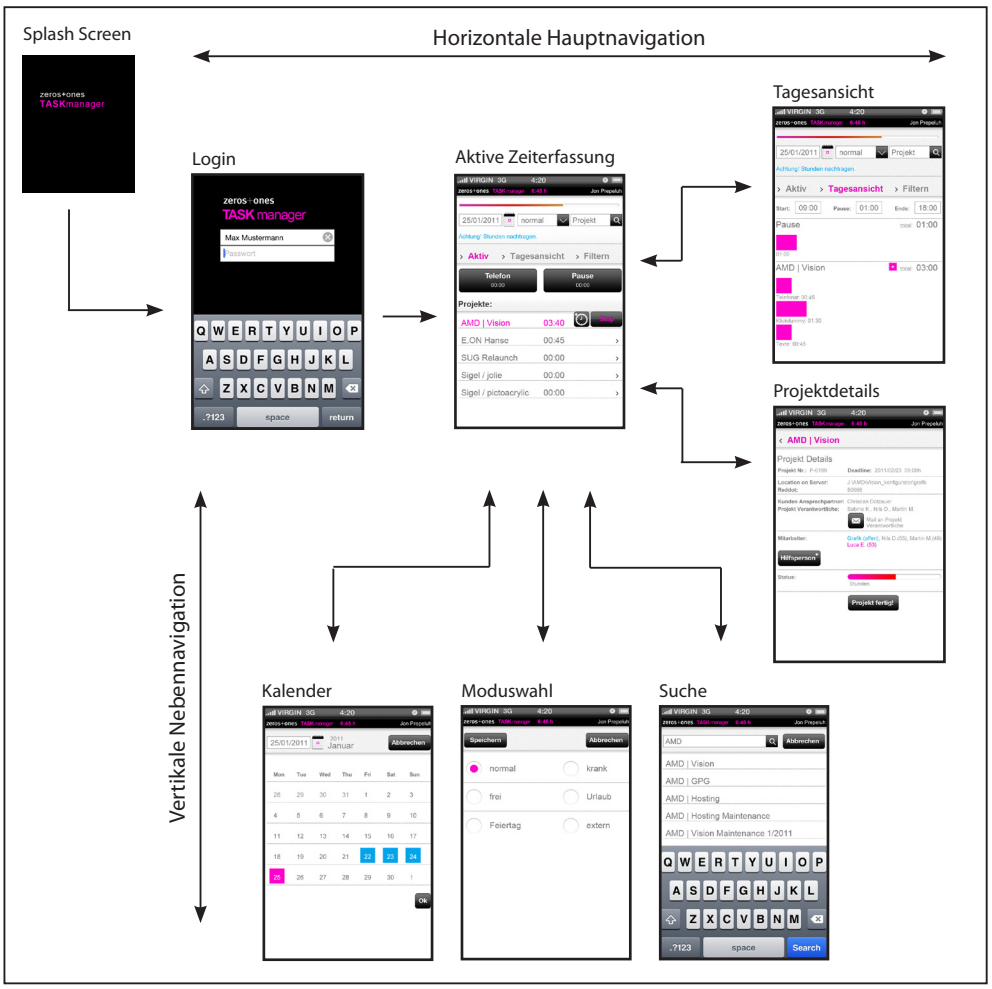


Abbildung 6.2: Schematische Übersicht der Zusammenhänge und Übergänge zwischen den einzelnen Views der Taskmanager-App

Grafiken und Daten umfangreicher Anwendungen müssen generell im Voraus geladen werden, bevor der erste Screen angezeigt werden kann. Bei mobilen Applikationen wird für die Dauer der Ladezeit oft ein Splash-Screen angezeigt. Sobald die App fertig geladen ist, wird dieser durch das tatsächliche UI ausgetauscht (vgl. Abbildung 6.2) [Corlan 2011]. Der Splash-Screen besteht aus nur einer Grafik und kann über die Eigenschaft `splashScreenImage` der ViewNavigatorApplication angegeben werden (vgl. Listing 3.3).

■ SPLASH-SCREEN: Grafischer Platzhalter während eines Boot- oder Ladevorgangs.

```
<s:ViewNavigatorApplication
   xmlns:fx="http://ns.adobe.com/mxml/2009"
   xmlns:s="library://ns.adobe.com/flex/spark"
   firstView="views.View_Login"
   initialize="initMain(event)"
   splashScreenImage="@Embed(,assets/splash.png')">
```

Listing 3.3: Taskmanager.mxml, splashScreenImage

Navigation und Datenübergabe

Die wichtigste Eigenschaft einer View-Komponente heißt `navigator`. Sie repräsentiert eine Instanz der ViewNavigator-Klasse, an welche die jeweilge View angebunden ist. Über die `navigator`-Eigenschaft können neue Views auf dem Stack abgelegt oder zur letzten View navigiert werden. [Corlan 2011]

```
navigator.pushView(SecondView, data)
navigator.popView();
navigator.popToFirstView();
navigator.popAll();
```

Listing 3.4: ViewNavigator.as, ausgewählte Methoden

Die in Listing 3.4 aufgeführten Methoden unterstützen das Navigieren zwischen einzelnen Views. Soll beispielsweise eine zweite View angezeigt werden, muss diese zunächst als neue Komponente vom Typ `spark.components.View` im Paket „views“ erstellt werden und kann anschließend durch die Methode `navigator.pushView(SecondView)` präsentiert werden. [Corlan 2011]

Jede View-Komponente besitzt eine Eigenschaft `data` vom Typ `Object`, welche die in der View benötigten Daten spezifiziert. Die Daten können dabei in einem beliebigen Format bereitgestellt werden. Für den Taskmanager wurden die Mitarbeiterprojekte beispielsweise im XML-Format aufbereitet und der `data`-Property zugewiesen. Die Daten können dann während der User-Navigation zwischen den Views weitergereicht werden. Zusätzlich ermöglicht die `data`-Property dem ViewNavigator, die View-Daten zwischenzuspeichern. [Jaramillo01 2011] Dabei bleiben die Werte sowohl beim Navigieren zwischen verschiedenen Views innerhalb derselben App bestehen, als auch beim Verlassen und erneuten Öffnen der App. Listing 3.4 zeigt die Übergabe der XML-Daten für die Mitarbeiterprojekte beim Einblenden der `View_Aktiv`. In Listing 3.5 ist der Eventhandler für das Ereignis `mx.events.FlexEvent.DATA_CHANGE` dargestellt. Hier kann nun auf die `data`-Property der View zugegriffen und diese als Quelle für die Projektliste genutzt werden. Die Projekte sind per Data-Binding an den Datenprovider der Liste und somit an die View gebunden und gehen auch beim Deaktivieren der View nicht verloren.

■ PROPERTY: Eigenschaft eines Objektes oder einer Klasse.

■ DATA-BINDING: Automatische Synchronisation der Daten durch Bindung dieser an ein bestimmes Objekt.

```
protected function but_login_clickHandler(e:MouseEvent)
{
  this.navigator.pushView(View_Aktiv, xml_projects);
}
```

Listing 3.5: View_Login.mxml, but_login_clickHandler()

```
protected function dataChangeHandler(e:FlexEvent)
{
  dp_projects.source = XML(data).children();
}
```

Listing 3.6: View_Aktiv.mxml, dataChangeHandler()

Über ein weiteres Argument kann ein bestimmter Übergangstyp eingestellt werden. Standardmäßig ist eine Wipe-Transition vorgegeben. Für alle pop-Aktionen wird ein Wipe von links nach rechts ausgeführt und für alle push-Aktionen ein Wipe von rechts nach links. [Tretola 2011] Dieses Verhalten hat sich bei mobilen Applikationen generell standardisiert. Wie in Abbildung 6.2 zu sehen, werden für die Taskmanager-App zwei verschiedene Übergangstypen eingesetzt. Für die Übergänge auf der horizontalen Hauptnavigationsachse wird die Standard-Transition verwendet. Da diese als Default-Wert eingestellt ist, muss sie nicht noch einmal als Argument übergeben werden. Listing 3.7 zeigt die Präsentation der Kalender-View im Taskmanager entlang der vertikalen Nebennavigationsachse von unten nach oben. Um von der Standard-Transition abweichende Überblendungen zu verwenden, müssen diese zunächst im `<fx:Declarations>` Block definiert werden und können dann der Methode `pushView()` mitgegeben werden.

■ WIPE:
Ein- und Ausblenden durch Wischen.

```
<fx:Declarations>
  <s:SlideViewTransition id="transSlideUP"
                      direction="up"/>
  <s:SlideViewTransition id="transSlideDOWN"
                      direction="down"/>
</fx:Declarations>

protected function icon_kalender_clickHandler
                      (event:MouseEvent):void
{
  navigator.pushView(View_Kalender, null,
                      null, transSlideUP);
}
```

Listing 3.7: View_Aktiv.mxml, dataChangeHandler()

Abbildung 6.3: iPhone 4: Auswahl-View nimmt nur halben Screen ein

Wird eine View auf den View-Stack „gepusht", nimmt diese immer den gesamten Screen ein. Mittels der `pushView()`-Methode ist es bisher nicht möglich eine View, nur bis zur Hälfte des Displays zu animieren, so dass die darunter liegende View weiterhin sichtbar bleibt. Da dies jedoch eine oft verwendete Technik (vgl. Abbildung 6.3) ist und auch in der Taskmanager-App für das Präsentieren der View für den ZeitDialog vorgesehen ist, wäre diese Möglichkeit wünschenswert.

■ HARDWARE-BUTTON: Im Gehäuse des Gerätes verankerte Buttons mit globaler Funktionalität.

Mobile Geräte verfügen über unterschiedliche Hardware-Buttons. iPhone und iPad haben jeweils nur einen einzigen runden Button, der den User jeder Zeit zurück zum Home-Screen navigiert, unabhängig von der aktuell gestarteten App. Android-Geräte dagegen werden meistens mit vier Hardware-Buttons ausgeliefert, ein Home-Button, ein Menü-Button, ein Zurück-Button sowie ein Button für die Suche (vgl. Abbilung 6.4). Die Funktionen der Hardware-Buttons unter Android können von jeder Android App genutzt und überschrieben werden.

Abbildung 6.4: Hardware-Buttons des iPhones (oben) und Android (unten)

Eine mit Flex erstellte mobile ViewNavigatorApplication verarbeitet die Events der Android-Hardware-Buttons „Zurück" und „Menü" bereits ohne zusätzliche Codezeilen. Wird in einer View-basierten Anwendung der „Zurück"-Button betätigt, führt Flex automatisch `navigator.popView()` aus und navigiert zur vorherigen View. Dies ist eine sehr elegante Lösung, da sich der Entwickler keine Gedanken über diese Android-spezifischen Unterschiede bei seiner Cross-Plattform App machen muss. Dennoch ist dieses Verhalten nicht immer wünschenswert. Beispielsweise ist es im Taskmanager nicht

gewollt, dass der Anwender nach dem erfolgreichen Login vom Screen der aktiven Zeiterfassung wieder zurück zum Login-Screen navigieren kann. Dies wäre jedoch durch das Tappen auf den „Zurück"-Button unter Android der Fall. Um diese Reaktion zu unterdrücken, kann im Eventhandler des `FlexEvent.BACK_KEY_PRESSED` der Befehl `preventDefault()` ausgeführt werden (vgl. Listing 3.8). Wird der „Menü"-Button eines Android-Gerätes gedrückt, erscheint das `<s:ViewMenu/>` einer View, falls dieses vom Entwickler definiert wurde.

```
this.addEventListener(FlexEvent.BACK_KEY_PRESSED,
                      onBackKeyPressed);
protected function onBackKeyPressed(e:FlexEvent):void
  {
        e.preventDefault();
  }
```

Listing 3.8: View_Aktiv.mxml, onBackKeyPressed()

View Lifecycle

■ VIEW LIFECYCLE: Lebenszyklus einer View-Komponente.

Da der Zwischenspeicher mobiler Endgeräte oft sehr begrenzt ist, wurde die View-Klasse um einige wichtige Events ergänzt, welche erkennen wann eine View angezeigt werden soll und wann sie wieder im Hintergrund verschwindet. Im Kapitel II wurden diese bereits kurz vorgestellt (vgl. Listing 1.5). Um Daten bis zur erneuten Aktivierung einer View aufzubewahren, können diese in der Property `View.data` gespeichert werden. Wird eine View durch die Aktivierung einer anderen View deaktiviert, so wird diese aufgrund der verfolgten Destruction-Policy (vgl. Abbildung 6.5) standardmäßig zerstört und alle Daten, die nicht in der `data`-Property gespeichert sind, gehen verloren. Dieses Prinzip ist vorteilhaft, um die Speicherbelegung minimal zu halten, macht es allerdings komplizierter, Daten View-übergreifend zu verwalten. [Tretola 2011]

■ DESTRUCTION-POLICY: Strategie des Erzeugens und Zerstörens von View-Objekten im Hinblick auf die Optimierung der Speicherbelegung.

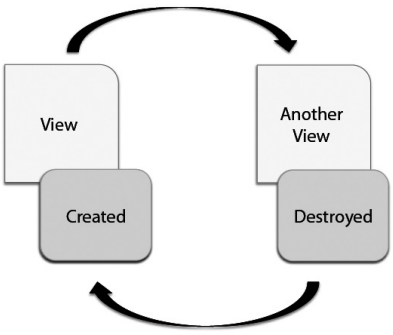


Abbildung 6.5: Flex SDK 4.5.1: View Lifecycle [Corlan 2011]

Um jedoch Werte zu erhalten, die nicht View-spezifisch, aber dennoch an die View-Darstellung gebunden sind, muss die Deaktivierung komplett unterbunden werden. Eingestellt werden kann

dies über das Setzten der Eigenschaft `destructionPolicy="never"` in der View-Deklaration. Für die Implementierung des Prototyps war diese Maßnahme erforderlich, um den Timer in der aktiven Zeiterfassung erhalten zu können, auch wenn der User zwischenzeitlich zur Tagesansicht wechselt.

Aufbau einer View

Eine View ist in Flex in zwei Bereiche aufgeteilt, ActionBar und View Content (vgl. Abbildung 6.6). Die ActionBar kann dabei View-übergreifend in der Haupteinstiegsklasse oder für jede einzelne View erneut definiert werden. Der View Content enthält die individuellen Komponenten, die die jeweilige View ausmachen.

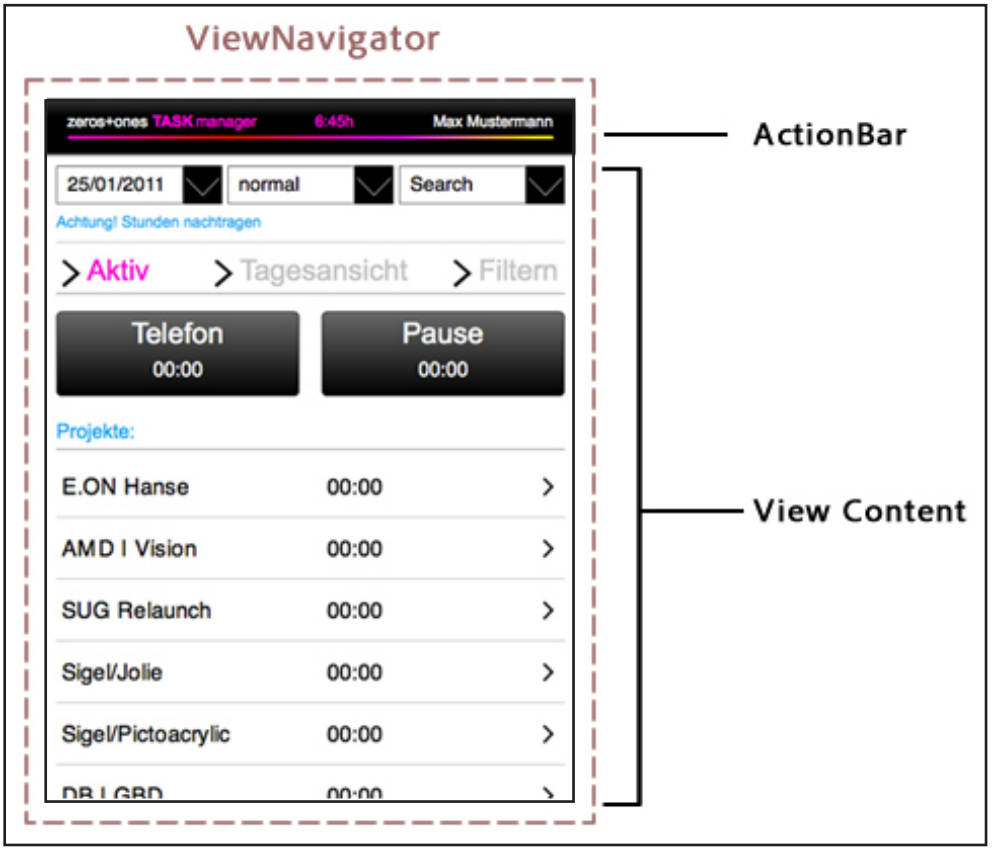


Abbildung 6.6: Flex SDK: Aufbau einer View [opensource.adobe02 2011]

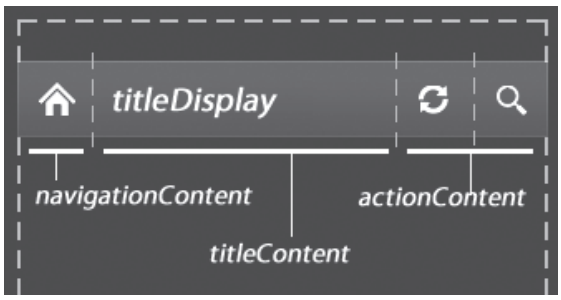


Abbildung 6.7: Flex SDK: Aufbau der ActionBar [opensource.adobe01 2011]

Die ActionBar dient der Verwaltung grundlegender Funktionen, die in der View benötigt werden. Dafür wurde die ActionBar in drei Funktionskerne unterteilt: Navigation, Titel und Aktionen. Im Navigationsbereich können zum Beispiel Buttons integriert werden, die auf den Home-Screen zurück navigieren. Der Titelbereich enthält oft die Bezeichnung der aktuellen View und im Aktionsbereich sind Buttons denkbar zum Starten einer Suche oder sonstiger Funktionen, die in der View benötigt werden. [Corlan 2011] Abbildung 6.7 zeigt eine mögliche Verwendung der drei Bereiche und Listing 3.9 die entsprechende Beschreibung in MXML.

```
<s:navigationContent>
  <s:Button icon="@Embed(,icons/home.png')"/>
</s:navigationContent>
<s:titleContent>
  <s:Label text="titleDisplay"/>
</s:titleContent>
<s:actionContent>
  <s:Button icon="@Embed(,icons/synch.png')"/>
  <s:Button icon="@Embed(,icons/search.png')"/>
</s:actionContent>
```

Listing 3.9: ActionBar Deklaration

Die ActionBar im Taskmanager soll laut Konzept keine Aktionen beinhalten, dafür enthält sie komplexen textuellen Inhalt (vgl. Abbildung 4.7). So sollen hier das Agenturlogo, die Loginzeit sowie der Mitarbeitername angezeigt werden (vgl. Abbildung 6.6). Zusätzlich wird die Motivations-Progressbar in die ActionBar integriert. Da sich die Darstellung der ActionBar im Navigationsverlauf nicht ändert, braucht diese nur einmal in der Einstiegsklasse angelegt werden. Ausnahme bildet die Login-View. Da die ActionBar hier noch nicht angezeigt werden soll, wird für diese View die Property `actionBarVisible = false` gesetzt.

Um alle Informationen konzeptgetreu abzubilden, werden Logo, Zeit, Name und Progressbar in einer separaten Komponente gruppiert und diese dem <s:titleContent> Tag zugeordnet (vgl. Listing 4.0). Die Breite des Titelbereichs wird auf 100% festgelegt, wodurch sich die Darstellung der ActionBar-Komponente über die gesamte Displaybreite erstreckt und die Bereiche für Navigation und Aktionen in der Taskmanager-ActionBar ignoriert werden.

```
<s:titleContent>
  <components:COMP_ActionbarContent width="100%"
   horizontalAlign="center" top="3" bottom="3"/>
</s:titleContent>
```

Listing 4.0: Taskmanager.mxml, ActionBar

III.2.2.3 Herausforderung Multi-Density

Neben der Vielfalt mobiler Betriebssysteme stellt auch die Masse mobiler Endgeräte, die bereits auf dem Markt verfügbar sind und noch sein werden, eine große Herausforderung für die Entwicklung dar. Eine Applikation muss sowohl für kleine Telefon-Screens als auch für große Tablet-Screens optimiert sein. Dies ist notwendig, da im App-Market prinzipiell keine Unterscheidung zwischen Phone- und Tablet-Apps gemacht wird. Eine App kann somit auf allen Geräten, die mit dem jeweiligen Betriebssystem ausgestattet sind, installiert werden. So soll auch die Taskmanager-App später beispielsweise aus dem Android-Market auf einem „HTC Desire" Handy (Android) wie auch auf einem „Samsung Galaxy Tab" (Android) installiert werden können. Ebenso soll sie sich aus Apples App-Store auf einem iPhone 4 (iOS) und einem iPad 2 (iOS) gleichermaßen installieren lassen und in ihrer Darstellung an die jeweilige Displayauflösung anpassen.

■ MULTI-DENSITY:
Density = Dichte. Die Anzahl der Pixel innerhalb der sichtbaren Screenfläche. Auch als DPI oder PPI bezeichnet. Ein Screen mit geringer DPI verfügt über weniger Pixel innerhalb einer bestimmten Fläche als ein Screen mit hoher DPI.

■ DPI:
Dots per Inch. Wird äquivalent zu PPI (Pixel per Inch) verwendet.

Daher ist ein wesentlicher Aspekt bei der Erstellung plattformunabhängiger Apps die Beachtung der verschiedenen Displaygrößen. Mobilgeräte können sowohl verschiedene Auflösungen (Breite x Höhe in Pixeln) als auch verschiedene DPIs (auch Density) haben (vgl. Tabelle 1.3). [Adobe07 2011]

Gerät	**Plattform**	**Auflösung**	**DPI**
Apple iPhone 4	iOS	640x960	326
Apple iPad 2	iOS	1024x768	132
HTC Desire	Android	480x800	252
Google Nexus One	Android	480x800	252
Motorola Droid	Android	480x854	265
Motorola Xoom	Android	1280x800	150
Blackberry Playbook	Blackberry Tablet OS	1024x600	167
Samsung Galaxy S	Android	480x800	233
Samsung Galaxy Tab	Android	1024x600	168

Tabelle 1.3: Beispiele für die unterschiedlichen Auflösungen und DPIs mobiler Geräte in Anlehnung an [Jose02 2011]

So haben beispielsweise die meisten Tablets und einige Telefone eine Pixeldichte von 160 DPI, während andere Telefone eine Pixeldichte von 240 oder 320 DPI haben. Dadurch werden Elemente mit fixen Pixelmaßen auf Geräten mit höherer Pixeldichte kleiner erscheinen. Besonders auf Touch-Interfaces kann dies zu Problemen führen, da Kontrollelemente stets groß genug sein müssen, um sie gut mit einem Finger bedienen zu können (vgl. Abbildung 7.1). [Jaramillo01 2011]

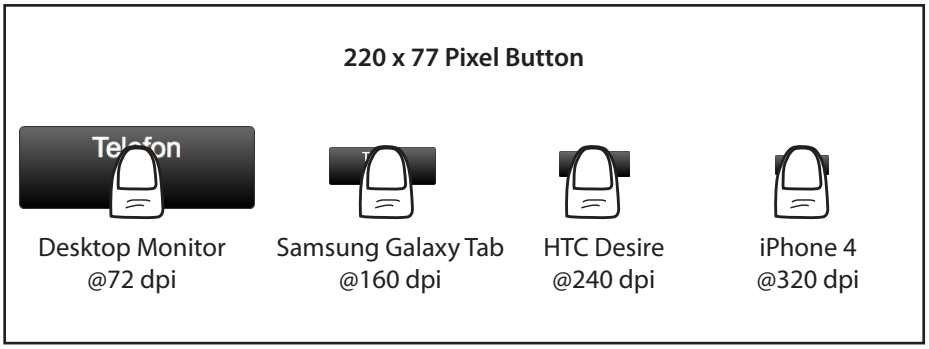


Abbildung 7.1: DPI-Problem: gleiche Pixelanzahl, verschiedene physikalische Maße, in Anlehnung an [Jaramillo02]

Adobe hat dem Flex SDK 4.5 wesentliche neue Funktionen zur Unterstützung mobiler Geräte mit verschiedenen Pixeldichten und Screenauflösungen hinzugefügt. Diese Features beinhalten unter anderem einen automatischen Skalierungsansatz, Multi-DPI Bitmaps, CSS-basierte Styles und DPI-spezifische Mobile-Skins. Prinzipiell kann der Entwickler zwischen zwei Ansätzen wählen: automatisches DPI-Management und manuelles DPI-Management. [Jose02 2011]

■ CSS: „Cascading Style Sheets" - dienen der Beschreibung von Stilvorlagen.

■ STYLES: Stilvorlagen. Beschreiben z. B. Farbe, Größe und Anordnung von UI-Elementen.

■ MOBILE-SKINS: Für kleine Displays optimierte Designs.

Automatisches DPI Management

Aufgrund der großen Anzahl vorkommender DPIs nimmt Flex eine Kategoriesierung in drei Klassen vor: 160 DPI Geräte, 240 DPI Geräte und 320 DPI Geräte. Smartphones mit 238 DPI oder 249 DPI werden beispielsweise beide der Klasse der 240 DPI Geräte zugeordnet (vgl. Tabelle 1.4). [Jaramillo02 2011]

Classification	160 DPI	240 DPI	320 DPI
Devices	Most tablets iPhone 3GS Motorola Droid Pro	Most Android phones	iPhone 4
Mapped range	< 200 DPI	>= 200 DPI <= 280 DPI	> 280 DPI
Typical range	132 DPI (iPad) to 181 DPI (HTC Hero)	217 DPI (HTC Evo) to 254 DPI (NexusOne)	326 DPI (iPhone 4)

Tabelle 1.4: Aufteilung der Geräte in DPI-Klassen [Jaramillo02 2011]

Auch Desktop-Monitore unterscheiden sich in ihren Screenauflösungen. Die Flex-Komponenten sind darum seit Beginn an für die Erstellung dynamischer Layouts konzipiert wurden. Vektorbasierte Layout-Elemente, wie `Group`, `VGroup` und `HGroup`, füllen durch Setzen der Elementbreite auf 100% stets die gesamte Breite des Screens, unabhängig davon ob die Auflösung 480x854 oder 600x786 beträgt. [Adobe07 2011]

■ GROUP: Flex-Komponente, die als Container zur Schachtelung weiterer Elemente fungiert.

Für Mobilgeräte reicht dieser herkömmliche Skalierungsansatz jedoch nicht mehr aus. Die zusätzlichen Unterschiede in der Pixeldichte erfordern eine zusätzliche Anpassung der Kontrollelemente. Zu diesem Zweck wurde im Flex SDK 4.5 eine neue Property eingeführt: `applicationDPI`. Mit Hilfe dieser Eigenschaft kann der Entwickler die DPI-Klasse festlegen, für die er das Layout der Anwendung optimiert hat. Wird zur Laufzeit eine von dieser DPI abweichende Pixeldichte festgestellt (`runtimeDPI`), wird die gesamte Anwendung um einen entsprechenden Faktor skaliert. Das Ergebnis ist eine App, die für eine spezifische DPI designt wurde und dennoch auf Geräten mit abweichender DPI einen guten Eindruck macht [Jose03 2011].

Ist applicationDPI beispielsweise auf 160 gesetzt und das Zielgerät hat ebenfalls eine `runtimeDPI` von 160, beträgt der Skalierungsfaktor 1 und es erfolgt keine Skalierung. Entspricht die `runtimeDPI` jedoch 240 DPI, wie zum Beispiel auf dem HTC Desire, so werden sämtliche Komponenten der Applikation um 150% skaliert. Wird die App auf einem 320 DPI Gerät ausgeführt, wie das iPhone 4, ergibt sich ein Skalierungsfaktor von 200%. [Jose02 2011] Abbildung 7.2 zeigt das Verhalten der Taskmanager-App bei einer automatischen Skalierung mit `applicationDPI = 160` (vgl. Listing 5.1).

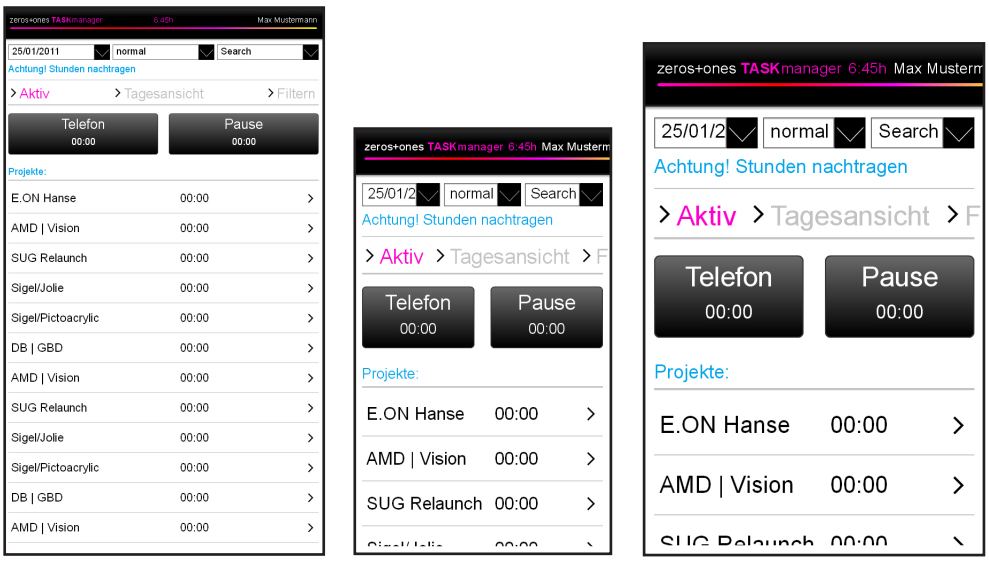


Abbildung 7.2: Automatische Skalierung der Taskmanager-App mit applicationDPI=160 und CSS Styles für DPI=160

```
<s:ViewNavigatorApplication
   ...
   firstView="views.View_Login"
   initialize="initMain(event)"
   splashScreenImage="@Embed(,assets/splash.png')"
   applicationDPI=160
>
```

Listing 5.1: Taskmanager.mxml, application=160

Empfehlenswert ist es, die Eigenschaft `applicationDPI` stets auf die kleinste DPI zu setzen, die unterstützt werden soll. Grund dafür ist, dass vektorbasierte Grafiken tendenziell besser aussehen und positioniert werden können, wenn sie vergrößert statt verkleinert werden. [Jaramillo01 2011]

Die für den mobilen Einsatz optimierten Flex-Komponenten passen sich bereits von Haus aus den verschiedenen DPIs an. Dabei werden abhängig von der gewählten `applicationDPI` entsprechende Assets geladen. Ist `applicationDPI` nicht spezifiziert, passen sich die Skins entsprechend der `runtimeDPI` an.

■ ASSETS: Sammlung von Grafikelementen, die in der Anwednung benötigt werden.

Durch die automatische Skalierung bei höher aufgelösten Displays werden auch alle verwendeten Bitmaps um den entsprechenden Faktor skaliert.

Pixelbasierte Grafiken erzeugen jedoch im Unterschied zu vektorbasierten Grafiken beim Vergrößern unschöne Artefakte. Um diesem Umstand entgegenzuwirken, sollte für jede der drei DPI-Klassen ein entsprechend aufgelöstes Bitmap zur Verfügung gestellt werden, welches dann zur Laufzeit abhängig von der registrierten `runtimeDPI` geladen werden kann.

Dieses Vorgehen wird durch die Einführung der Klasse `MultiDPIBitmapSource` seit Flex 4.5 erleichtert. Die Klasse verfügt über drei Eigenschaften: `source160dpi`, `source240dpi` und `source320dpi`. [Jose02 2011] Sie kann bei allen Komponenten eingesetzt werden, die Grafiken einbetten können. Beispielsweise in der `Image`-Komponente (vgl. Listing 5.2) oder als Icon-Property für Buttons. [Jaramillo01 2011]

```
<s:Image>
  <s:source>
   <s:MultiDPIBitmapSource
         source160dpi="assets/Logo160.png"
         source240dpi="assets/Logo240.png"
         source320dpi="assets/Logo320.png"/>
  </s:source>
</s:Image>
```

Listing 5.2: View_Login.mxml, MultiDPIBitmapSource Logo

Auch Fonts werden beim automatischen Skalierungsansatz vergrößert. Anstatt jedoch Texte über ihre Vektoren zu skalieren, wählt Flex automatisch eine größere Schriftgröße, so dass hier zusätzlicher Aufwand erspart bleibt. [Jose02 2011]

■ FONT: Schriftfamilie.

Manuelles DPI Management

Wird die Property `applicationDPI` nicht gesetzt, so wird das automatische Skalieren unterbunden und manuelles Skalierungsmanagement wird ermöglicht. Die Eigenschaft wird beim Initialisieren der Anwendung durch Flex auf die detektierte `runtimeDPI` gesetzt. Es ist nun die Aufgabe des Entwicklers, für jede der drei DPI-Klassen entsprechende Skalierungsvorgaben festzulegen. Die Built-in Komponenten des Flex SDKs passen sich dennoch automatisch an die jeweilige Pixeldichte an, indem sie die passenden Assets laden. Auch für die Skalierung von Bitmaps kann auf die bereits erwähnte Klasse `MultiDPIBitmapSource` zurückgegriffen werden, die sich für das manuelle Skalieren ebenso eignet wie für das automatische.

Für alle anderen Layouts und Skins müssen jedoch separate Regeln festgelegt werden. Im Speziellen sollten für jedes Element pro DPI-Klasse eigene Font-Größen und Abstandsmaße definiert werden. Hierfür wurde dem Flex SDK 4.5 der Support von CSS Media Queries hinzugefügt, die es ermöglichen, konkrete CSS-Regeln zu definieren, die nur für bestimmte DPIs gelten. Wird nun ein Gerät mit 240 DPI erkannt, verwendet die App automatisch die Styles, welche in der verknüpften CSS-Datei unter `@media (application-dpi:240)` definiert wurden (vgl. Listing 5.3).

■ CSS MEDIA QUERIES: Ermöglichen das Hinzufügen von Styles zur Laufzeit basierend auf DPI und/oder Plattform des Zielgerätes.

```
/*Every os-platform @ 240dpi */
@media (application-dpi:240)
{
   .labelTitlebar
   {
    fontSize: 14;
   }
   .labelTextInputLogin
   {
    fontSize: 22;
   }
   .labelTextInputGlobals
   {
    fontSize: 18;
   }

   ...
}
```

Listing 5.3: Taskmanager.css, Auszug

Flex 4.5 unterstützt ein zusätzliches Attribut innerhalb des `@media` Tags: `os-platform`. Mit Hilfe dieser Eigenschaft können plattformspezifische Styles definiert werden. Beispielsweise kann der Flex `ActionBar` unter Android und iOS eine unterschiedliche `chromeColor` zugewiesen und so ein spezielles Aussehen verliehen werden (vgl. Listing 5.4). [Jose01 2011]

■ CHROME COLOR: Glanzfarbe der Komponente.

```
ActionBar {chromeColor:#000000;}
@media (os-platform: „Android“)
{
   ActionBar { chromeColor: #999999; /* dark gray */ }
}
@media (os-platform: „IOS“)
{
   ActionBar { chromeColor: #6DA482; /* blue */ }
}
```

Listing 5.4: @media (os-platform) ActionBar [Jose01 2011]

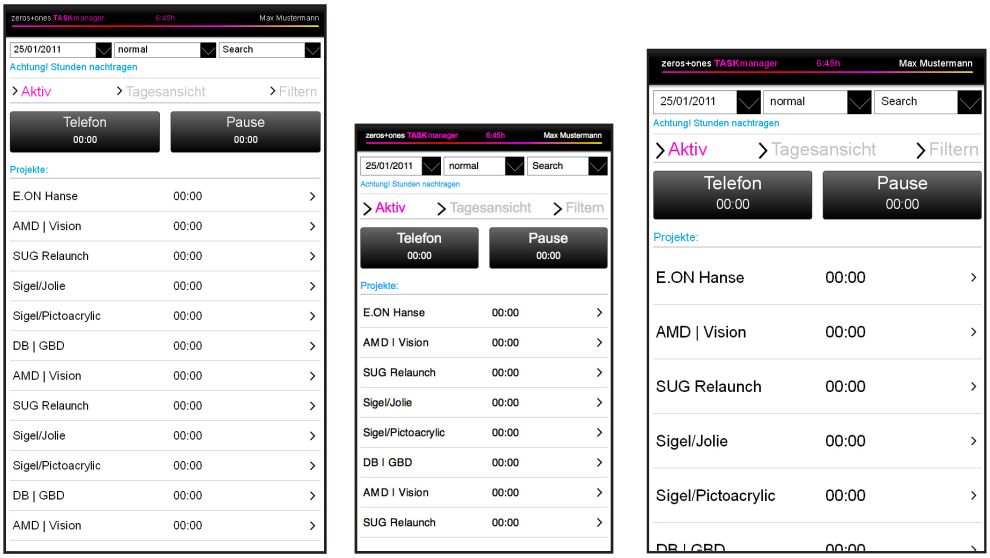
Einige Eigenschaften können nicht über CSS-Styles angegeben werden. Um diese dennoch zur Laufzeit gezielt auszuwählen, können jederzeit durch die `applicationDPI` bedingte Anweisungen festgelegt und ausgeführt werden. Für die Taskmanager-App wurde dieses Prinzip zur Anpassung der `ActionBar`-Höhe verwendet (vgl. Listing 5.5).

```
switch(applicationDPI)
{
   case 160:
    this.actionBar.height = 60;
   break;

   case 240:
    this.actionBar.height = 50;
   break;

   case 320:
    this.actionBar.height = 70;
   break;
}
```

Listing 5.5: Taskmanager.mxml, creationCompleteHandler

Durch Verwendung der neu eingeführten Eigenschaften `applicationDPI`, `MultiDPIBitmapSource` und `@media` CSS-Styles kann eine App ihre Darstellung an eine Vielzahl unterschiedlicher Displayauflösungen anpassen, sogar an Größen und Pixeldichten, die zum Zeitpunkt der Entwicklung noch gar nicht auf dem Markt waren [Goldman 2011].

Die Auswirkungen des manuellen DPI Managements auf die Darstellung des Taskmanagers auf den verschiedenen Testgeräten sind in Abbildung 7.3 abgebildet.



Samsung Galaxy Tab	HTC Desire	iPhone 4
1024 x 600	480 x 800	960 x 640
@168 DPI	@252 DPI	@326 DPI
Klasse: 160 DPI	Klasse: 240 DPI	Klasse: 320 DPI

Abbildung 7.3: Manuelle Skalierung der Taskmanager-App, applicationDPI ist nicht gesetzt, CSS Styles für DPI=160, 240, 320

Vergleicht man Abbildung 7.2 und Abbildung 7.3 wird deutlich, welcher der beiden Sklaierungsansätze für die Taskmanager-App geeigneter ist. Die automatische Skalierung liefert zwar ein akzeptables Ergebnis, erfordert jedoch noch einige manuelle Korrekturen. So werden viele Texte zwar auf höher aufgelösten Displays entsprechend vergrößert, erscheinen dann aber schnell zu groß oder werden abgeschnitten. Durch eine manuelle Definition der Schriftgrößen kann die Positionierung und Darstellung der Textfelder und Komponenten sehr viel effizienter vorgegeben werden.

III.3 Veröffentlichung

Wie bereits im Kapitel III.2.1 erwähnt, unterstützt die Flash Builder IDE 4.5 zum aktuellen Zeitpunkt die Erstellung von Adobe AIR Applikationen für folgende Plattformen:

Desktopsysteme

- Microsoft Windows
- Mac OS (Apple)
- Linux

Mobilsysteme

- Android (Google)
- iOS (Apple)
- Blackberry Tablet OS (RIM)

Von den im Kapitel II.2.3.3 aufgestellten mobilen Zielplattformen für die Taskmanager-App werden dementsprechend drei von vier Systemen abgedeckt. Einzig der Export für Smartphones mit dem Betriebssystem „Windows Phone 7“ ist momentan noch nicht integriert.

Der grundsätzliche Exportvorgang zum Publizieren einer App für den Market ist für jedes der einzelnen Systeme ähnlich und erfolgt wie auch für den Desktop über den Menüpunkt „Export Release Build“. Der Hauptunterschied beim Export eines Release Builds für mobile Plattformen ist, dass Flash Builder die Anwendung als native App packt und nicht als .air-File, wie das für den Desktop der Fall ist (vgl. Abbildung 8.1). Für Android wird beispielsweise ein .apk-File erzeugt, welches genauso aussieht wie eine native Android App. [Jaramillo01 2011] Für iOS wird ein .ipa-File und für Blackberry ein .bar-File erzeugt. Der Prozess zur Erstellung von Provisioning Profilen und den benötigten Zertifikaten zum Signieren der App unterscheidet sich für die verschiedenen Plattformen grundlegend und soll im Folgenden kurz erläutert werden [Stallons01 2011].

■ RELEASE BUILD:
Publizierbare Version der Anwendung.

■ PROVISIONING PROFILE:
Eine Datei, die es erlaubt Apps während der Entwicklung auf einem Testgerät zu installieren.

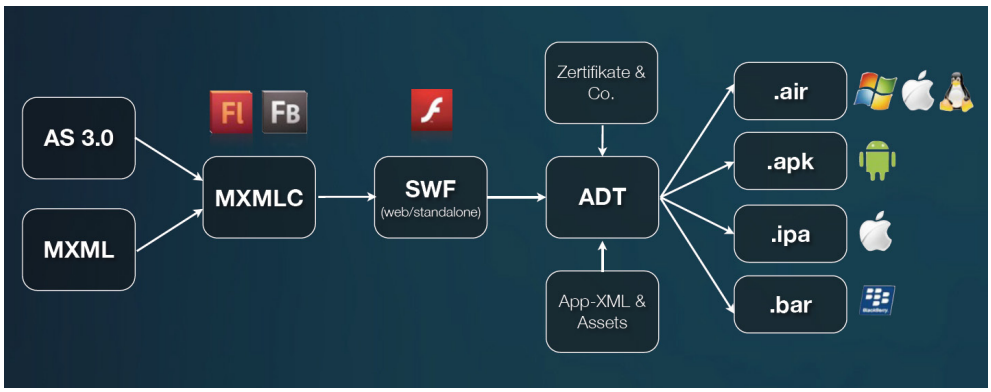


Abbildung 8.1: Ablauf der Veröffentlichung [Ünlü 2011]

Android

Der größte Unterschied beim Erstellen von AIR Apps für Android im Vergleich zu iOS oder Blackberry ist, dass diese auch ohne Signatur live über USB auf einem Gerät getestet und debuggt werden können. Erst für die Veröffentlichung der App für den Android Market muss eine digitale Signatur hinzugefügt werden.

Eine Signatur hat dabei zwei Ziele. Zum einen versichert sie dem User, dass der Entwickler der App vertrauenswürdig ist und die Anwendung keinen bösartigen Code auf dem Endgerät ausführen wird. Zum anderen kann über eine Signatur sicher gestellt werden, dass die App seit dem Veröffentlichungszeitpunkt nicht geändert wurde. Um eine zertifizierte Signatur für Android zu erstellen gibt es zwei Möglichkeiten. Mit Hilfe des Flash Builders kann entweder ein „self-signed" Zertifikat erstellt werden oder man bezieht es offiziell über eine autorisierte Zertifizierungsstelle. Zum Veröffentlichen reicht jedoch eine selbst erstellte Signatur aus. [Stallons01 2011]

■ SELF SIGNED:
Ein selbst signiertes Zertifikat wird von seinem eigenen Ersteller unterzeichnet.

Bevor die Anwendung publiziert wird, sollten die erforderlichen Android-Einstellungen und -Berechtigungen in der Deskriptor-XML vorgenommen werden. Nachdem die erstellte Signatur zugewiesen wurde, kann das APK File erzeugt und in den Android Market hochgeladen werden. Sollte die AIR Runtime noch nicht auf dem Endgerät installiert sein, wird diese beim erstmaligen Ausführen der App automatisch aus dem Market bezogen [Stallons01 2011].

■ APK:
Android Package Format.

Der Aufwand für den gesamten Exportprozess ist verhältnismäßig gering. Alle nötigen Schritte können mit Hilfe des Flash Builders getätigt werden.

Die App lässt sich schnell und ohne weitere Umstände auf einem Android-Gerät testen und debuggen.

iOS

Um während der Entwicklung einer iPhone-Anwendung seine Ergebnisse bereits auf einem Device ausführen und testen zu können, bedarf es einiger Voraussetzungen. Im Gegensatz zu Android wird bereits für das live-debugging ein Zertifikat sowie ein „Provisioning Profile" benötigt. Ohne diese Files kann eine AIR Anwendung für iOS ausschließlich im Emulator getestet werden.

Bezogen werden können die Files über das Apple Developer Portal, nachdem man sich dort gegen eine Jahresgebühr von $99 als Entwickler registriert hat. Apple müssen nun über die folgenden Schritte die App-ID, das Entwicklungsgerät (z.B. MacBook), sowie das zum Testen verwendete iPhone oder iPad bekannt gemacht werden. Aus diesen drei Informationen wird ein Profil erstellt, welches das Debuggen dieser konkreten App, mit dem angegebenem Rechner und dem zugewiesenem iPhone gestattet. Das Vorgehen ist folgendermaßen:

- Mit Hilfe des Mac OS Dienstprogrammes „Schlüsselbundverwaltung" ist ein Entwicklungszertifikat zu erstellen
- Dieses muss im Developer Provisioning Portal unter der Rubrik „Certificates" als neues Zertifikat hochgeladen werden
- Nachdem es durch den Verantwortlichen verifiziert wurde, kann das generierte „Code Signing Certificate" heruntergeladen und auf dem Entwicklungsrechner installiert werden
- Unter der Rubrik „App IDs" muss nun eine App ID vergeben werden. Es bietet sich an, eine wildcard bundle ID zu erstellen, die für mehrere Applikationen verwendet werden kann
- Unter der Rubrik „Devices" können nun über die UDID mehrere Testgeräte hinzugefügt werden
- Im letzten Schritt werden unter dem Menüpunkt „Provisioning" die benötigten Ressourcen zu einem „Provisioning Profile" zusammengesetzt

■ WILDCARD BUNDLE ID: Eine App-ID, die mit Platzhalter erstellte wird, so dass diese für mehrere Apps einsetzbar ist.

■ UDID: „Unique Device Identifier" eines iOS-Gerätes. Besteht aus ca. 40 Zeichen.

- Das erstellte „Provisioning Profile“ kann heruntergeladen und zusammen mit dem erstellten Developer Zertifikat in den Flash Builder Einstellungen verknüpft werden

Will man eine native iPhone-Anwendung mit der dafür vorgesehenen Entwicklungsumgebung XCode programmieren, so ist man mangels der Unterstützung anderer Betriebssysteme an Mac OS gebunden. Mit der Flash Builder IDE können jedoch auch unter Windows Apps für iPhone und iPad erstellt und getestet werden.

Beim Veröffentlichen der App als Release oder Debug Version für iOS wird ein .ipa-File erstellt. Der Exportvorgang kann dabei mehrere Minuten dauern, was den Entwicklungs-Workflow erheblich beeinträchtigt. Im Unterschied zu Android wird das erzeugte .ipa-File nicht direkt auf das angeschlossene iPhone oder iPad überspielt, sondern muss zunächst mit der Apple Synchronisations-Software „iTunes“ geöffnet und über die verknüpfte Medienbibliothek des Gerätes synchronisiert werden. Dieser Prozess ist sehr umständlich und macht häufiges Debuggen zu einem komplizierten und teuren Vorgang.

Anders als bei Android und Blackberry Tablet OS erfordert iOS keine externe AIR Runtime. Wie bereits im Kapitel II.3.2 erörtert, verfolgt Adobe unter iOS einen generativen Cross-Plattform Ansatz. Aus dem Sourcecode wird ein natives Binärfile kompiliert, welches ohne Laufzeitumgebung auskommt (vgl. Abbildung 8.2).

Generell lässt sich sagen, dass sich auch bei der Verwendung von Cross-Plattform Frameworks der aufwendige Zertifizierungsprozess von Apple

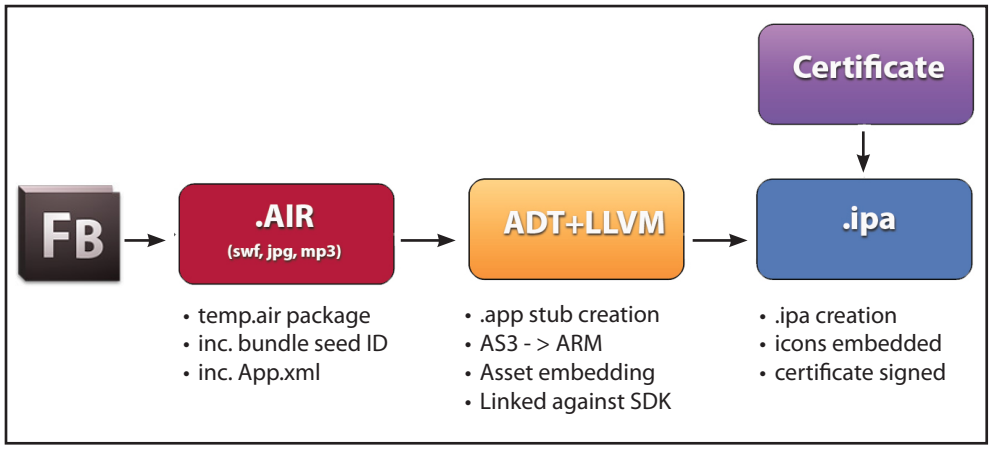


Abbildung 8.2: Flash Builder 4.5 Packager Workflow für iOS Apps, in Anlehnung an [Duvos 2010]

nicht vermeiden lässt. Der zusätzliche Umweg beim Aufspielen einer App auf ein Testgerät über iTunes macht den Entwicklungsprozess für iOS etwas umständlich.

Blackberry Tablet OS

Auch für die Zielplattform Blackberry Tablet OS, die bisher nur auf dem Blackberry Playbook verwendet wird, wird bereits für das live-debugging ein Zertifikat benötigt. Dieses kann kostenfrei vom Hersteller bezogen und anschließend im Flash Builder registriert werden. Nachdem auch das Testgerät mit einem sogenannten Debug Token ausgestattet wurde und sämtliche benötigte Einstellungen in der Konfigurationsdatei „blackberry-tablet.xml" vorgenommen wurden, kann die App live auf einem über USB verbundenen PlayBook getestet werden.

■ DEBUG TOKEN: Bringt App mit dem Testgerät in Verbindung.

Ohne Testgerät kann die App ausschließlich emuliert werden. Hierfür bieten sich dem Entwickler jedoch zwei Varianten. Zum einen kann der interne Emulator des Flash Builders verwendet werden, zum anderen stellt Blackberry einen eigenen Playbook Simulator mit einigen Zusatzfunktionen zur Verfügung. Um diesen Simulator nutzen zu können, sind folgende Schritte erforderlich [nach Stallons02 2011]:

- Download und Installieren des Blackberry Tablet OS SDK
- Registrieren des Blackberry Tablet OS SDK Pfades im Flash Builder
- Download und Installieren der Software „VMWare"
- Installieren des Blackberry Playbook Simulators in VMWare
- IP Adresse des Simulators beziehen
- Simulator als Testgerät im Flash Builder registrieren

Die AIR Runtime ist auf Geräten mit dem Blackberry Tablet OS bereits vorinstalliert, so dass im Grunde keine Maßnahmen getroffen werden müssen, diese nachzuinstallieren. Der gesamte Test- und Debug-Vorgang im Simulator ist relativ einfach zu bewerkstelligen. Leider konnte die Taskmanager-App mangels Verfügbarkeit nicht auf einem realen Blackberry Playbook getestet werden.

Desktop Widget

Obwohl die Taskmanager-App im Flash Builder als „Mobiles Flex Projekt“ angelegt wurde, ermöglicht Adobe dennoch die Kompilierung der Sourcen als Desktop AIR Anwendung. Dies erfolgt wiederum über den Menübefehl „Export Release Build“, wobei nun „Signed AIR package for installation on Deskop“ ausgewählt werden muss. Auch AIR Applikationen für den Desktop müssen signiert werden. Dazu kann, wie auch bei Android Apps, selbstständig ein Zertifikat mit Hilfe der IDE erstellt oder über eine autorisierte Zertifizierungsstelle erworben werden. Nachdem die Signatur hinzugefügt wurde, erstellt Flash Builder das AIR-Paket „Taskmanager.air“, welches anschließend auf jedem Rechner installiert werden kann. Auch auf Desktopsystemen ist die AIR Runtime Voraussetzung für das Installieren und Ausführen von AIR Anwendungen.

Die Widget-Version des Taskmanagers ähnelt zunächst aufgrund der typischen Systemtaskleiste eher einem normalen Desktop-Programm (vgl. Abbildung 8.3). Dies kann in der Konfigurationsdatei „Taskmanager-app.xml“ sehr einfach angepasst werden:

```
<initialWindow>
   <systemChrome>none</systemChrome>
   <transparent>true</transparent>
</initialWindow>
```

Listing 5.6: Taskmanager-app.xml, Widget-Einstellungen

Um dem Widget einige Zusatzfunktionen sowie einen transparenten Rahmen zu geben (vgl. Abbildung 8.4), bietet es sich an, ein neues Projekt anzulegen, welches auf die Sourcen des mobilen Taskmanager-Projektes zugreift. Dort können zusätzliche CSS-Styles und Einstellungen ergänzt werden.

Der Aufwand für die Erstellung einer Desktop-Variante hängt maßgeblich von den Unterschieden und Erweiterungen zur mobilen Version ab. Ein eigenes Desktop-Projekt anzulegen ist prinzipiell nicht notwendig.

Abbildung 8.3: Taskmanager-App als Desktop-Programm mit Taskleiste

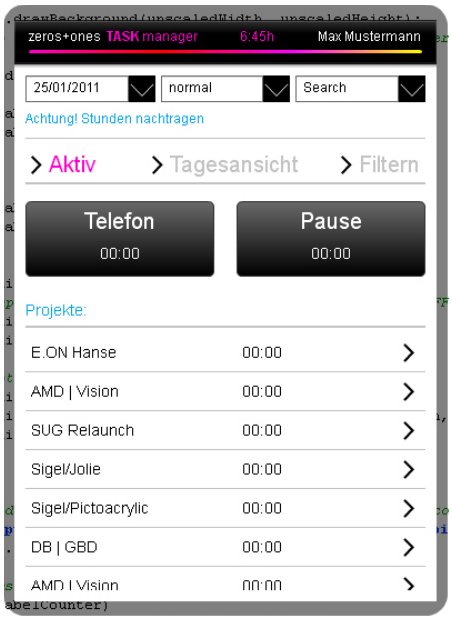


Abbildung 8.4: Taskmanager-App als Desktop-Widget ohne Taskleiste

IV Fazit und Ausblick

Mit dieser Diplomarbeit wurde die Verwendbarkeit von Adobe AIR als Cross-Plattform Laufzeitumgebung für mobile Applikationen untersucht und dokumentiert. Auf Grundlage der aktuellen Marktsituation wurden hierfür vier Mobilsysteme als Zielplattformen für die Portierung ausgewählt: Android, iOS, Windows Phone 7 und Blackberry Tablet OS. Nachdem die mit der Adobe Integrated Runtime verfolgte Cross-Plattform Strategie herausgestellt wurde, konnten im Rahmen der prototypischen Umsetzung einer Taskmanager-App konkrete Schwerpunkte der mobilen Interface- und Interaktions-Programmierung auf ihre Realisierbarkeit untersucht werden.

Die Entwicklung mobiler Systeme und Frameworks erfolgte in den letzten Jahren mit wachsender Geschwindigkeit. So war noch zu Beginn der Recherchen zu diesem Thema im März 2011 die Adobe Flash Builder IDE nur als Beta Version unter dem Entwicklungsnamen „Burrito“ erhältlich und Android war das einzige unterstützte Mobilsystem. Auch nach dem ersten Release im Mai 2011 konnten die neuen mobilen Features nur begrenzt für iPhone und iPad implementiert werden. Erst im Juni 2011 wurde der vollständige Support von iOS sowie Blackberry Tablet OS in die IDE integriert.

So verlief der Arbeitsprozess gewissermaßen parallel mit der Entwicklung Adobes am AIR SDK. Durch diesen Umstand mussten die Erkenntnisse immer wieder neu an den Milestones von AIR und Flash Builder orientiert werden. Für das Ergebnis dieser Arbeit hatte dies dennoch überwiegend positive Auswirkungen, denn am Ende konnten dadurch mehr Askpekte in die Untersuchung einbezogen werden. Schon für Ende 2011 hat Adobe das nächste Update für die Flash Builder IDE angekündigt. Die Möglichkeit für eine Fortsetzung der bisherigen Überlegungen ist also gegeben.

IV.1 Ergebnisse und Bewertung

Wie bereits im Kapitel III.3 zusammengefasst, wird durch die Flash Builder IDE bisher die Erstellung und Veröffentlichung von Applikationen für drei mobile Zielplattformen unterstützt. Dazu zählen die für die Diplomarbeit festgelegten Systeme Android und iOS sowie Blackberry Tablet OS. Eine Unterstützung von Windows Phone 7 wurde bisher nicht integriert, was im Hinblick auf die wachsenden Marktanteile der Plattform jedoch wünschenswert wäre.

Die Anzahl der durch das AIR SDK angebundenen Systemschnittstellen ist reichlich. Tabelle 6.8 verdeutlicht, dass grundlegende mobile Features wie Multitouch-Erkennung, GPS und Beschleunigungssensorik implementierbar sind. Native Erweiterungen, wie der schreibende Zugriff auf Kalender- und Adressbuchdaten, stehen jedoch noch aus. Auch Bluetooth und Vibrationssensor können bisher nicht mit einer AIR App angesprochen werden. Mit der Weiterentwicklung mobiler Geräte werden immer mehr Hardware-Features hinzukommen, deren Funktionen eventuell nie oder erst sehr viel später in das AIR SDK integriert werden. Für den Einsatz AIRs als Cross-Plattform Framework bedeutet dies einen großen Nachteil, da die technischen Möglichkeiten immer denen der nativen Umgebung hinterherhängen werden. Obwohl die bisher ins SDK aufgenommenen API-Funktionen gegenüber den noch fehlenden eindeutig überwiegen, braucht es für dieses Problem zukünftig eine Lösung.

Ergebnisse der praktischen Untersuchung

Die schwerpunktbezogene praktische Untersuchung hielt viele positive Beobachtungen bezüglich der Aufbereitung des Flex und AIR SDKs für die Entwicklung mobiler AIR Applikationen bereit. Diese sowie die beobachteten Nachteile werden im Folgenden zusammengefasst.

Adobe Flash Builder 4.5 ermöglicht dem Entwickler eine unkomplizierte und übersichtliche Projektstrukturierung. Dabei ist für die Programmierung einer mobilen AIR App für alle drei Zielplattformen lediglich ein einziges Projekt anzulegen. Soll die Anwendung zusätzlich als Desktop- oder Web-Applikation veröffentlicht werden, können die gemeinsamen Funkti-

onen und Datenanbindungen in einem Library-Projekt gekapselt werden, welches als externe Bibliothek in die UI-Projekte inkludiert wird. Der Projekttyp „Library-Projekt“ schafft viel Transparenz und ermöglicht eine modularisierte Verwaltung der Projekte.

Für den Support von Multitouch-Gesten wurde das AIR SDK um neue Eventklassen zum Erkennen von Touch-Eingaben ergänzt. Einfache Desktop-Anwendungen können ohne weitere Anpassungen für ein mobiles Gerät aufbereitet werden, da die verwendeten Mouse Events automatisch auf Single Touch Events abgebildet werden. Als vorteilhaft erweist sich außerdem der automatische Support des onscreen keyboards durch alle Textverarbeitungskomponenten.

Besonders hilfreich bei der Erstellung mobiler Anwendungen ist die Möglichkeit, spezielle Verhaltensweisen – wie automatische Screenausrichtung oder das Bildseitenverhältnis der App – über die zentrale Konfigurations-XML festlegen zu können. Auch die Zugriffsberechtigungen einer App können hier definiert werden, zumindest für die Zielplattform Android. Nachteilig ist, dass selbige Berechtigungen für Blackberry nochmals in einer separaten Konfigurationsdatei vorgenommen werden müssen, die beim Export des .bar-Pakets Voraussetzung ist. Für iOS ist die Festlegung der Zugriffe nicht notwendig.

Als problematisch hat sich auch die Unterstützung von Multitasking auf den verschiedenen Systemen herausgestellt. Eine für die Taskmanager-App essentielle Timerfunktion läuft unter Android auch bei deaktivierter App im Hintergrund weiter, während sie unter iOS unterbrochen wird, so dass hierfür ein Workaround gefunden werden musste. Das Multitasking-Modell von iPhone und iPad wird durch AIR aktuell nicht unterstützt, wäre aber für die Umsetzung des Taskmanagers sehr vorteilhaft gewesen. Unter Blackberry Tablet OS konnte der Multitasking-Support in Ermangelung eines realen Gerätes nicht getestet werden.

Sehr gelungen ist dem Flex-Team die Integration der mobilen Screen Metapher. Über drei zur Auswahl stehende Templates kann auf einfachste Weise von dem View-basierten Navigationsmodell Gebrauch gemacht werden. Verankerte Push- und Pop-Methoden sowie definierbare Übergangsanimationen erleichtern die Navigation und den Datenaustausch zwischen

einzelnen Views. Der Aufbau einer Standard-View ist sinnvoll aufgeteilt und flexibel gestaltbar. Die neue ActionBar-Komponente verfügt über die gängigen Funktionen einer modernen Navigationsleiste auf mobilen Systemen. Besonders vorteilhaft wirkt sich die automatische Einbindung der Hardware-Buttons unter Android und Blackberry Tablet OS aus. Durch die integrierte Destruction-Policy bietet Flex außerdem ein optimiertes Speichermanagement.

Beim Erstellen neuer Views innerhalb der App ist es bisher leider nicht möglich, andere Kontrollelemente als Subviews hinzuzufügen, welche beispielsweise nur Teile des Screens verdecken. Dieses Prinzip ist jedoch sowohl bei nativen Android als auch iOS Apps gängig und wäre daher auch unter Flex wünschenswert.

Eine der größten Herausforderungen bei der plattformübergreifenden Entwicklung stellt die Vielzahl auftretender Screen-DPIs dar. Um diese zu bewältigen, wurde das aktuelle Flex SDK gleich um mehrere Lösungsmöglichkeiten erweitert. Der Entwickler hat die Wahl zwischen automatischem DPI Management, wobei sämtliche Skalierungen vom System übernommen werden, oder manuellem DPI Management. Mittels CSS können DPI-abhängige Styles festgelegt werden und neue Klassen ermöglichen das Hinterlegen verschieden aufgelöster Bitmap-Grafiken. Eine große Menge der herkömmlichen Flex-Komponenten wurde bereits für den mobilen Einsatz optimiert. Dies beinhaltet unter anderem die dynamische Anpassung der jeweiligen Komponenten-Skins an die entsprechende Display-DPI.

Konnte die Entwicklung für die verschiedenen Zielplattformen relativ einheitlich erfolgen, so sind für die Veröffentlichung teilweise stark unterschiedliche und aufwendige Prozesse nötig. Eher kompliziert stellen sich die für Release- und Debug-Version vorausgesetzten Zertifizierungsverfahren für Blackberry Tablet OS und vor allem iOS dar. Diese, von den jeweiligen Herstellern vorgegebenen, Schritte sind auch bei der Erstellung nativer Apps erforderlich und können daher von Adobe nicht umgangen werden. AIR Apps für Android dagegen sind ohne Signatur live auf einem Testgerät ausführbar und können sogar mit einer selbst erstellten Signatur publiziert werden.

Große Probleme ergaben sich beim Testen und Debuggen der Taskmanager-App für iOS-Geräte. Im Gegensatz zu Android kann der Exportvorgang hier mehrere Minuten beanspruchen und die Ausführung auf dem Testgerät ist ausschließlich über die Zusatzsoftware iTunes möglich. Dieses Prozedere beeinträchtigt den Entwicklungs-Workflow stark und wird hoffentlich über kommende Updates optimiert.

Ein Vorteil des Flash Builders gegenüber der nativen iOS-Entwicklungsumgebung XCode ist, dass dieser auch unter Microsoft Windows iPhone Apps exportieren kann. Beim Erstellen der Release Builds erzeugt Flash Builder für jede Plattform native App-Pakete, die im jeweiligen Market vertrieben werden können. Auch die Erstellung einer Desktop-App aus einem mobilen Projekt heraus ist möglich. Auffällig ist die verhältnismäßig große Dateigröße der generierten iOS App (vgl. Tabelle 1.5). Grund dafür ist, dass die App auf den anderen Systemen durch die AIR Runtime ausgeführt wird und daher relativ klein verpackt werden kann. Dies ist aus den bereits aufgeführten Gründen unter iOS nicht möglich, weshalb der ActionScript Sourcecode samt Laufzeitumgebung in nativen iOS-Bytecode kompiliert wird, was die Datei um ca. 6 MB vergrößert.

	Desktop	**Android**	**iOS**	**Blackberry Tablet OS**
Dateigröße	758 KB	786 KB	7,2 MB	758 KB

Tabelle 1.5: Vergleich Dateigröße der paketierten App

AIR als mobile Cross-Plattform Lösung

Adobe AIR wurde von Beginn an als Cross-Plattform Umgebung konzipiert. Eine einmalig mit ActionScript oder Flex entwickelte Anwendung läuft unter den Desktopsystemen Windows, Linux und Mac OS. Seit 2011 hat Adobe die Runtime nun auch für mobile Plattformen aufbereitet. Die Untersuchungen im Rahmen der Diplomarbeit haben gezeigt, dass es mit Hilfe der Flash Builder IDE 4.5 bereits möglich ist, eine mobile App aus nur einer Codebasis heraus für die Mobilsysteme Android, iOS und Blackberry Tablet OS zu veröffentlichen.

Grundsätzlich konnten die Hauptanforderungen der Taskmanager-App mit Adobe AIR beziehungsweise Flex und ActionScript umgesetzt werden. Wie im praktischen Teil der Arbeit beschrieben, mussten für einige Probleme Fallunterscheidungen vorgenommen oder Workarounds gefunden werden, was jedoch der prinzipiellen Umsetzbarkeit nicht im Wege stand. Der mobile Entwicklungs-Workflow ist durch die Erweiterungen des Flash Builders intuitiv vorgegeben und hält eine Reihe von Templates bereit. Mobile Besonderheiten wie das Navigieren zwischen Views oder die Anpassung des Layouts an die zahlreichen verschiedenen Screen-Auflösungen und -DPIs wurden erfolgreich integriert. Zudem konnte auf einfachste Weise zusätzlich zur Adressierung von drei verschiedenen Mobilplattformen eine Desktop-Variante der App erzeugt werden. Zusammenfassend war Adobe AIR als plattformübergreifende Lösung für die Umsetzung des Prototyps geeignet.

Wie jedes Cross-Plattform Framework ist jedoch auch Adobe AIR nur eine auf das native System aufgesetzte Schicht. Daher kann eine AIR App immer nur einen Kompromiss in User-Experience und Performance im Vergleich zu nativen Applikationen bilden. Dennoch hat der Prototyp insgesamt auf den einzelnen Systemen einen relativ performanten Eindruck gemacht. Dabei wurde festgestellt, dass die Performance nicht primär vom Betriebssystem abhängt, sondern vielmehr von der Leistung des Endgerätes.

So wie es mit Sicherheit zahlreiche App-Konzepte gibt, für die Adobe AIR aufgrund der genannten Einschränkungen nicht in Frage kommt, so gibt es dennoch Anwendungsfälle, bei denen AIR die erste Wahl sein kann.

Demnach ist es generell sehr geeignet, je einfacher die App aufgebaut ist und um möglichst schnell viele Plattformen abzudecken. Besonders gut lassen sich zum Beispiel Spiele mit der AIR Runtime umsetzen, da diese oft statt der standardmäßigen UI-Elemente individuelle Bitmap-Grafiken beinhalten. Der Unterschied zu einer nativen App lässt sich hier vom User nicht mehr wahrnehmen. Dabei sind kleine, weniger CPU-intensive Spiele aufgrund der Performance-Einschränkungen prädestinierter als umfangreiche 3D-Multiuser-Games.

Einen enormen Vorteil bietet die mobile Erweiterung von AIR, um bereits entwickelte Flash- und Flex-Anwendungen auf Mobilsysteme zu übertragen. Ein sehr erfolgreiches Beispiel ist das Flash Game „Machinarium“ von Amanita Design[10], welches sich lange im Web großer Beliebtheit erfreute und nun dank der neuen Möglichkeiten des AIR SDKs zusätzlich als iPad Spiel herausgebracht wurde. Mittlerweile ist es das erfolgreichste bezahlte iPad Spiel im App Store.

Für sehr komplexe App-Konzepte, die beispielsweise aufgrund hoher Qualitäts- und Sicherheitsansprüche prinzipiell nativ entwickelt werden sollten, eignet sich AIR auch als Prototyping-Umgebung. So können im Vorfeld der nativen Umsetzung Verhalten und Realisierbarkeit der App schnell und kostengünstig auf verschiedenen Plattformen getestet werden.

Auf Basis des momentanen Entwicklungsstands der Runtime lässt sich AIR nicht uneingeschränkt als Cross-Plattform Lösung empfehlen. Mit jeder hinzukommenden Zielplattform, Erweiterung der API-Funktionen und Performance-Verbesserungen würde es sich jedoch zunehmend qualifizieren.

Tablle 1.6 fasst weitere Vor- und Nachteile bezüglich der Verwendung von Adobe AIR zusammen.

10 http://amanita-design.net/ und http://www.youtube.com/watch?v=hVw4SVaopzw&feature=player_embedded

Vorteile:	mit nur einem Sourcecode können bereits drei mobile Systeme adressiert werden (+ Desktopsysteme, Web, TV)
	bei vorhandenen AS3/Flex Kenntnissen sehr schnelle Entwicklung möglich
	UI-Kontrollelemente sind einheitlich und auf allen Systemen gleich implementiert
	das Framework ist sehr ausführlich dokumentiert
	Adobe hat langjährige Erfahrung in der Cross-Plattform Entwicklung
	Wiederverwendung bestehender Desktop- u. Web-Projekte möglich
	AIR ist für das Blackberry Tablet OS favorisiertes App-Format
	mit dem Adobe Flash Builder 4.5 können iOS Apps unter Windows entwickelt, getestet und veröffentlicht werden
Nachteile:	keine nativen UI-Kontrollelemente integriert -> natives look-and-Feel kann nicht erreicht werden
	unterstützte API-Funktionen sind limitiert
	Performance liegt meistens unter der nativer Apps
	trotz einer Code-Basis dennoch Fallunterscheidungen notwendig (z. B. Multitasking)
	Dateigröße für iOS vergleichsweise groß

Tabelle 1.6: Weitere Vor- und Nachteile von Adobe AIR als mobile Laufzeitumgebung

Einordnung von AIR in mögliche Lösungsstrategien

In welchen Fällen Adobe AIR als mobile Laufzeitumgebung geeignet ist, lässt sich nicht pauschalisieren. Die Entscheidung hängt immer maßgeblich von den jeweiligen Anforderungen der zu entwickelnden App ab und sollte für jedes Projekt aufs Neue entschieden werden. Cross-Plattform Frameworks wie Adobe AIR kommen oft zum Einsatz, um Entwicklungszeit und -kosten gering zu halten. Dies kann für Auftraggeber und Entwickler gleichermaßen von Vorteil sein. Sind zum Beispiel später Erweiterungen oder Anpassungen an einer App vorzunehmen, muss dies nur einmal erfolgen und nicht einzeln für jede Zielplattform.

Dennoch sollte die Entscheidung nicht vorrangig von Kosten und Aufwand abhängig gemacht werden, sondern vielmehr im Sinne des Nutzers erfolgen. Das bedeutet, dass dem Auftraggeber die Einschränkungen von Adobe AIR als plattformübergreifende Lösung bekannt sein sollten. Dies betrifft insbesondere das erreichbare look-and-feel, welches aufgrund der fehlenden nativen UI-Kontrollelemente nicht mit dem einer nativ entwi-

ckelten App vergleichbar ist. Aber auch die Limitierungen der API-Schnittstellen und eventuelle Performance-Beeinträchtigungen sollten genannt werden.

Abbildung 8.5 sowie die nachfolgende Tabelle zeigen eine für die Agentur „Zeros + Ones“ aufgestellte Anforderungsanalyse sowie eine Entscheidungsmatrix zum Finden der passenden Entwicklungsstrategie für mobile Applikationen. Allgemein siedeln sich die Cross-Plattform Frameworks zwischen einer einfachen Lösung als mobile Website und einer teuren nativen Lösungsvariante an. Auf Grundlage der Untersuchungsergebnisse lässt sich Adobe AIR momentan eher den Frameworks für „Light Content“ zu ordnen. Damit sind einfache Apps gemeint, die vorwiegend aus 2D-Grafiken bestehen oder in einem informativen Kontext auf das Screen Metapher Prinzip aufbauen. „Rich Content“ wie Video und Geräte-Features können zwar problemlos integriert werden, leiden aber noch häufig unter Performance-Einbußen.

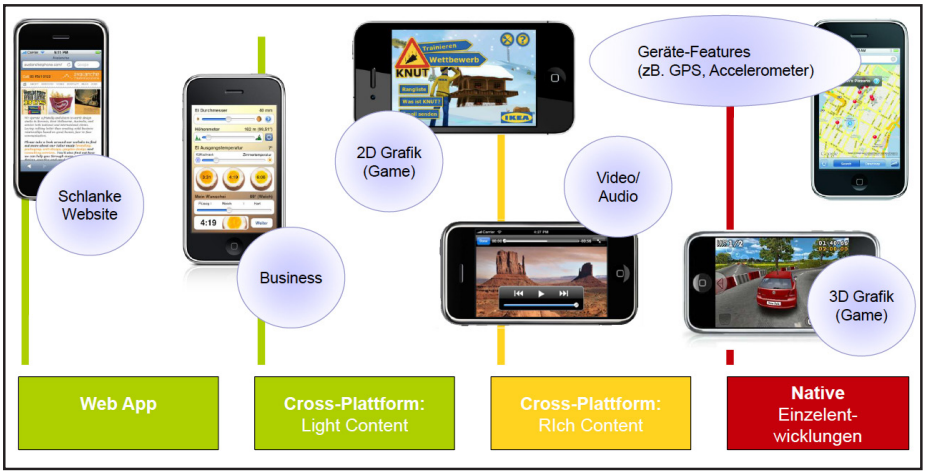


Abbildung 8.5: Anforderungsanalyse mobiler Applikationen

	Web App	**Cross-Plattform** Light Content	**Cross-Plattform** Rich Content	**Native App**
Entwicklungs-aufwand	wie Website	Wie Website +Markt +Hardware	Gering bei Vorkenntnissen in Technologie	Sehr hoch
Kosten pro App	wie Website	Wie Website +Markt +Hardware	Für X OS: Kosten für 1 App	Für X OS: X * Kosten für 1 App
App Market	NEIN	JA	JA	JA
Unterstützte Geräte-Features	Sehr wenig	Weniger	Fast alle	Alle
User-Experience	(semi-) NATIV	(semi-) NATIV	(semi-) NATIV	NATIV
Offline Verfügbarkeit	JA	JA	JA	JA

Tabelle 1.7: Entscheidungsmatrix für mobile Entwicklungsstrategie

IV.2 Ausblick

Die Erweiterung des AIR SDKs für mobile Systeme steht erst am Anfang. Laut Adobe soll das nächste Update der AIR Runtime auf die Version 3.0 sowie der neue Flash Player 11 bereits Anfang Oktober 2011 veröffentlicht werden. Unter den zahlreichen neuen Features, die bereits für dieses Release angekündigt wurden, wird es die Möglichkeit geben, AIR Apps um nativen Programm-Code zu erweitern [Adobe08 2011]. Mit Hilfe dieser sogenannten „Native Extensions" stellen fehlende API-Funktionen im AIR SDK kein Probelm mehr dar, denn entsprechende Schnittstellen lassen sich dann über Einbindung nativer Funktionsaufrufe ansprechen. Für eine App mit integrierter Bezahlfunktion kann so beispielsweise für jede Zielplattform ein angeschlossenes Kreditkartenlesegerät eingebunden werden. Native Erweiterungsmöglichkeiten bilden einen großen Vorteil für die Laufzeitumgebung, da somit jedes neue Geräte-Feature integriert werden kann, ohne auf ein Update der Runtime warten zu müssen.

Auch die Handhabung des Multitasking-Modells unter iOS wird mit dem Update viele Verbesserungen erfahren. So sollen AIR Apps zukünftig auch auf dem iPhone Audiodateien im Hintergrund abspielen können, wenn sie durch ein anderes Programm vorübergehend deaktiviert werden. Für die Taskmanager-App müsste folglich geprüft werden, ob der beschriebene Workaround für die Integration der Timerfunktion unter iOS weiterhin nötig ist oder dieses Problem mit dem Update bereits behoben werden kann.

Um AIR langfristig auch für reichhaltige Inhalte wie Video und 3D-Grafik zu qualifizieren, soll noch dieses Jahr die neue Stage3D-Architektur für AIR Apps eingeführt werden. Damit wird es möglich sein, die Rendergeschwindigkeit durch Hardwarebeschleunigung um ein tausendfaches zu beschleunigen [Nguyen 2011]. Auf dieser Grundlage könnte im Anschluß an diese Arbeit eine weitere Untersuchung feststellen, ob sich Adobe AIR zukünftig im Entscheidungsprozess auch für „Rich Content" Apps empfehlen lässt. Gerade für die plattformübergreifende Entwicklung komplexer 3D-Spiele wäre dies von großer Bedeutung.

Zudem wird sich die Anzahl der adressierbaren Systeme vergrößern. So sollen „weitere [mobile] Plattformen wie Windows Phone 7, das Palm webOS

und auch TV-Systeme wie Samsung SmartTV [...] voraussichtlich noch in diesem Jahr [ergänzt]“ werden [Widjaja02 2011]. Nach dem Update könnte somit auch das für den Taskmanager geforderte vierte Mobilsystem durch AIR abgedeckt werden und damit die drei meistverbreitetsten Plattformen weltweit: Android, iOS und Windows Phone 7.

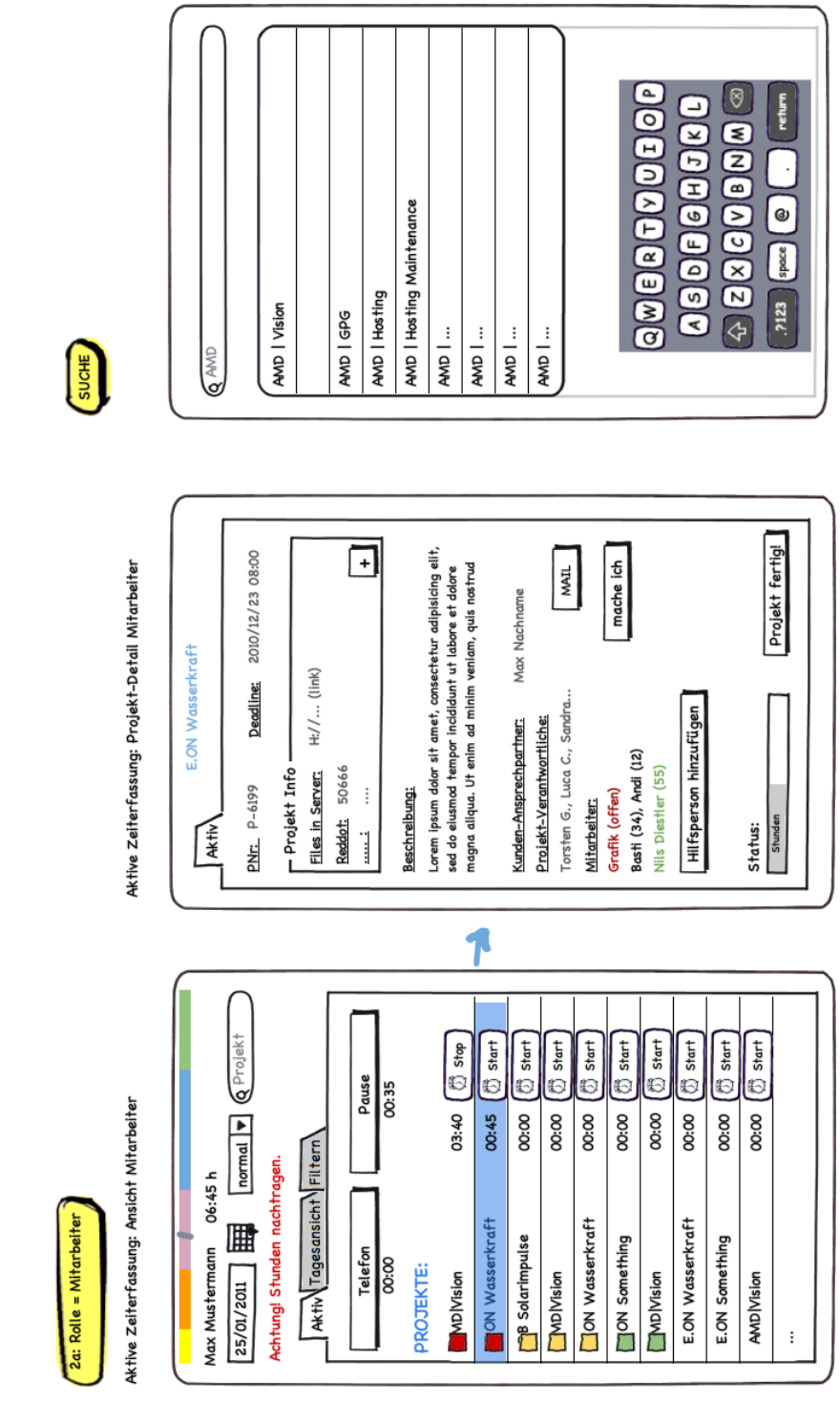
Dies ist nur eine Auswahl der geplanten Erweiterungen, die auf den Prerelease Seiten Adobes[11] nachzulesen sind. Adobe wird weiterhin signifikante Investitionen in die Entwicklung von Flex, AIR und der Flash Builder IDE vornehmen [Shorten02 2011]. Die allgemeine Einstellung Adobes im Bezug auf die Weiterentwicklung der mobilen Runtime ist für alle AIR-Entwickler durchaus motivierend. Sollten die geplanten Updates wie angekündigt erfolgen, bewegt sich Adobe AIR damit in eine sehr gute Ausgangslage gegenüber alternativen Cross-Plattform Framworks.

> „We’re continuing to focus on runtime performance, native extensions, new components, declarative skinning, adding more platforms and improving tooling workflows, such that in our next major release timeframe we expect that the need to build a fully-native application will be reserved for a small number of use cases“ [Shorten02 2011].

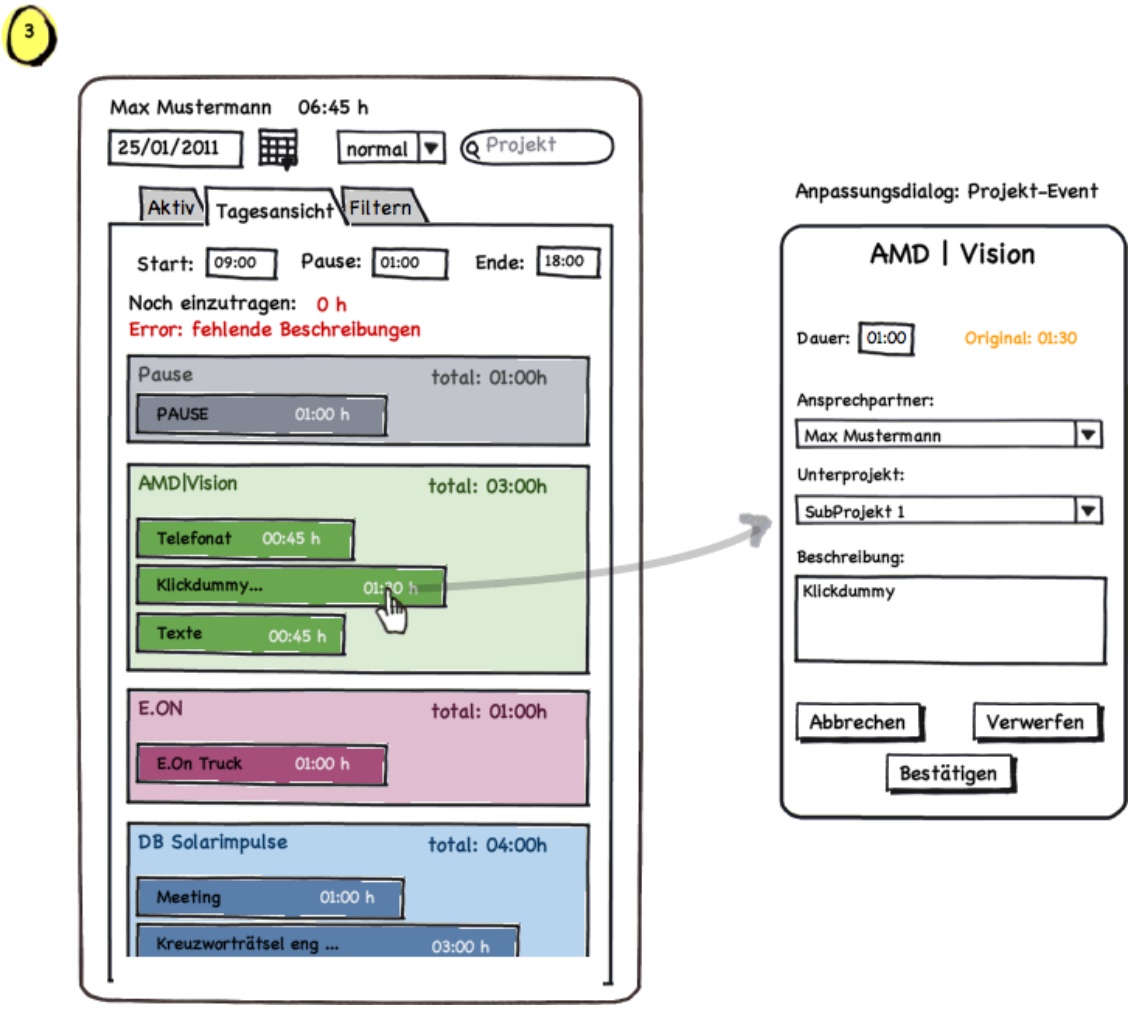
11 http://labs.adobe.com/technologies/flashplatformruntimes/air3/

A Anhang

A.1 Taskmanager – Mockups



Tagesansicht



A.2 Taskmanager – Design

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Literaturverzeichnis

[Adobe01 2010] Adobe. Adobe Unveils AIR on Mobile Devices; Readies Flash Player 10.1 for Launch. http://www.adobe.com/aboutadobe/pressroom/pressreleases/201002/021510FlashPlayerMWC.html. Zuletzt geprüft am 26.05.2011.

[Adobe02 2011] Adobe. Rich-Internet-Applikationen | Adobe AIR. http://www.adobe.com/de/products/air/. Zuletzt geprüft am 25.05.2011.

[Adobe03 2011] Adobe. Adobe AIR and Adobe Flash Player Incubator Features. http://labs.adobe.com/technologies/flashplatformruntimes/incubator/features/. Zuletzt geprüft am 27.05.2011.

[Adobe04 2011] Adobe. Adobe - Flash Player Statistics. http://www.adobe.com/products/player_census/flashplayer/. Zuletzt geprüft am 20.06.2011.

[Adobe05 2010] Adobe. Adobe Flash Builder 4.5 * About Flash Builder. http://help.adobe.com/en_US/flashbuilder/using/flashbuilder_4.5_help.pdf. Zuletzt geprüft am 20.07.2011.

[Adobe06 2011] Adobe. ActionScript® 3.0 Reference for the Adobe® Flash® Platform - ViewNavigatorApplication. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/ViewNavigatorApplication.html. Zuletzt geprüft am 02.08.2011.

[Adobe07 2011] Adobe. Developing Mobile Applications with ADOBE® FLEX® and ADOBE® FLASH® BUILDER. http://help.adobe.com/en_US/flex/mobileapps/developing_mobile_apps_flex.pdf. Zuletzt geprüft am 16.08.2011.

[Adobe08 2011] Adobe. Adobe AIR 3 Release Candidate. http://labs.adobe.com/technologies/flashplatformruntimes/air3/. Zuletzt geprüft am 22.09.2011.

[Allen 2010] Allen, Sarah/ Graupera, Vidal/ Lundrigan, Lee (2010): Pro smartphone cross-platform development. Springer Science + Business Media. New York.

[Apple 2011] Apple, Inc. App Store Review Guidelines - App Store Resource Center. https://developer.apple.com/appstore/resources/approval/guidelines.html. Zuletzt geprüft am 09.06.2011.

[Bansod 2011] Bansod, Aditya. Developing for iOS using Flash Professional | Adobe Developer Connection. http://www.adobe.com/devnet/logged_in/abansod_iphone.html. Zuletzt geprüft am 09.06.2011.

[Behrens 2010] Behrens, Heiko. Cross-Platform App Development for iPhone, Android & Co. http://heikobehrens.net/2010/10/11/cross-platform-app-development-for-iphone-android-co-%E2%80%94-a-comparison-i-presented-at-mobiletech-con-2010/. Zuletzt geprüft am 16.05.2011.

[Cantrell 2009] Cantrell, Christian. Multitouch and gesture support on the Flash Platform. http://www.adobe.com/devnet/flash/articles/multitouch_gestures.html. Zuletzt geprüft am 14.06.2011.

[Chambers 2010] Chambers, Mike. Flash Player 10.1 and Windows Phone 7. http://www.mikechambers.com/blog/2010/03/09/flash-player-10-1-and-windows-phone-7/. Zuletzt geprüft am 21.06.2011.

[Corlan 2011] Corlan, Mihai. Understanding Flex Mobile View and ViewNavigator. http://corlan.org/2011/01/12/understanding-flex-mobile-views-and-viewnavigator/. Zuletzt geprüft am 11.07.2011.

[Duvos 2010] Duvos, Enrique. RIAPT birthday party!!! https://acrobat.com/app.html#d=FEl7SiONtr-4x6yvzjw9fg. Zuletzt geprüft am 29.08.2011.

[Ehrenstein 2009] Ehrenstein, Constantin (2009): Adobe AIR. Galileo Press. Bonn.

[Elrom 2009] Elrom, Elad/ Janousek, Scott/ Joos, Thomas (2009): Flash on Devices. Friends of ED. Berkeley.

[Fling 2009] Fling, Brian (2009): Mobile design and development. O'Reilly. Beijing.

[Gartner 2011] Gartner. Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012. http://www.gartner.com/it/page.jsp?id=1622614. Zuletzt geprüft am 28.04.2011.

[GestureWorks 2011] GestureWorks. Multitouch Move, Rotate and Scale Tutorial. http://gestureworks.com/flash-tutorials/move-rotate-scale/. Zuletzt geprüft am 14.06.2011.

[Goldman 2011] Goldman, Oliver. Considerations for developing Adobe AIR applications for mobile. http://www.adobe.com/devnet/air/articles/considerations-air-apps-mobile.html. Zuletzt geprüft am 02.08.2011.

[Hochstätter 2011] Hochstätter, Christoph. Gartner: Knapp 50 Prozent Marktanteil für Android bis 2012. http://www.zdnet.de/news/mobile_wirtschaft_gartner_knapp_50_prozent_marktanteil_fuer_android_bis_2012_story-39002365-41551444-1.htm. Zuletzt geprüft am 20.04.2011.

[Jaramillo01 2011] Jaramillo, Narciso. Mobile development using Adobe Flex 4.5 SDK and Flash Builder 4.5. http://www.adobe.com/devnet/flex/articles/mobile-development-flex-flashbuilder.html. Zuletzt geprüft am 19.07.2011.

[Jaramillo02 2011] Jaramillo, Narciso. much ado about something. http://www.rictus.com/muchado/http://www.rictus.com/muchado/wp-content/uploads/2011/04/multiscreen-dev-with-flex-360flex-2011.pptx.pdf. Zuletzt geprüft am 16.08.2011.

[Jobs 2010] Jobs, Steve. Thoughts on Flash. http://www.apple.com/hotnews/thoughts-on-flash/. Zuletzt geprüft am 16.05.2011.

[Jose01 2011] Jose, Jason. Flex mobile skins – Part 3: Multiplatform development. http://macromediastudio.biz/devnet/flex/articles/mobile-skinning-part3.html. Zuletzt geprüft am 22.08.2011.

[Jose02 2011] Jose, Jason. Flex mobile skins – Part 2: Handling different pixel densities. http://www.adobe.com/devnet/flex/articles/mobile-skinning-part2.html. Zuletzt geprüft am 16.08.2011.

[Jose03 2011] Jose, Jason. Comparing CSS Media Queries vs. Application Scaling. http://blogs.adobe.com/jasonsj/2011/05/comparing-css-media-queries-vs-application-scaling.html. Zuletzt geprüft am 23.08.2011.

[Klingler 2011] Klingler, Anita. Gartner: iOS wird Tablet-Markt bis 2015 dominieren. http://www.zdnet.de/news/wirtschaft_unternehmen_business_gartner_ios_wird_tablet_markt_bis_2015_dominieren_story-39001020-41551626-1.htm. Zuletzt geprüft am 28.04.2011.

[Knor 2011] Knor, Max. Windows Phone 7 - Tipps & Tricks. Mobile Technology, 03 (2011, S. 46.)

[Leber 2011] Leber, Jürgen. Qual oder Wahl? Mobile Technology, 03 (2011, S.66.)

[Leggett 2006] Leggett, Richard/ Boer, Weyert de/ Janoušek, Scott (2006): Foundation Flash applications for mobile devices. Friends of ED. Berkeley, Calif.

[Nguyen 2011] Nguyen, Tom. Announcing Flash Player 11 and AIR 3. http://blogs.adobe.com/flashplatform/2011/09/announcing-flash-player-11-and-air-3.html. Zuletzt geprüft am 22.09.2011.

[opensource.adobe01 2011] opensource.adobe. Mobile ActionBar. http://opensource.adobe.com/wiki/display/flexsdk/Mobile+ActionBar. Zuletzt geprüft am 16.08.2011.

[opensource.adobe02 2011] opensource.adobe. View and ViewNavigator. http://opensource.adobe.com/wiki/display/flexsdk/View+and+ViewNavigator. Zuletzt geprüft am 16.08.2011.

[Shorten01 2011] Shorten, Andrew. Flex SDK and Flash Builder updates available – adds iOS and BlackBerry PlayBook support. http://blogs.adobe.com/flex/2011/06/flex-sdk-and-flash-builder-updates-available-adds-ios-and-blackberry-playbook-support.html. Zuletzt geprüft am 19.07.2011.

[Shorten02 2011] Shorten, Andrew. Flex: where we are headed. http://blogs.adobe.com/flex/2011/08/flex-where-were-headed.html. Zuletzt geprüft am 22.09.2011.

[Sonnati 2011] Sonnati, Fabio. More informations about Flash Player 11, AIR 2.7 and above. http://sonnati.wordpress.com/2011/04/20/more-informations-about-flash-player-11-air-2-7-and-above/. Zuletzt geprüft am 03.08.2011.

[Stallons01 2011] Stallons, Jeanette. Packaging applications for Google Android devices. http://www.adobe.com/devnet/air/articles/packaging-air-apps-android.html. Zuletzt geprüft am 29.08.2011.

[Stallons02 2011] Stallons, Jeanette. Packaging applications for BlackBerry Tablet OS devices. http://www.adobe.com/devnet/air/articles/packaging-air-apps-blackberry.html. Zuletzt geprüft am 29.08.2011.

[Subramaniam 2011] Subramaniam, Deepa. Flex 4.5 SDK, Flash Builder 4.5 and Flash Catalyst CS 5.5 Now Available! http://blogs.adobe.com/flex/2011/05/flex-4-5-sdk-flash-builder-4-5-and-flash-catalyst-cs-5-5-now-available.html. Zuletzt geprüft am 20.07.2011.

[taz 2011] taz. Kooperation bei Smartphones: Nokia macht's mit Microsoft. http://taz.de/1/zukunft/wirtschaft/artikel/1/nokia-machts-mit-microsoft/. Zuletzt geprüft am 27.04.2011.

[TECHNISM TODAY 2011] TECHNISM TODAY. Samsung Galaxy Tab vs Apple iPad. http://technismtoday.blogspot.com/2010/12/see-samsungs-up-and-coming-android.html. Zuletzt geprüft am 08.06.2011.

[Tretola 2011] Tretola, Rich (2011): Developing Android Applications With Flex 4.5. Oreilly & Associates Inc.

[typopark 2010] typopark. Android Betriebssystem Informationen. http://www.mobile-betriebssysteme.de/android.html. Zuletzt geprüft am 16.05.2011.

[Ünlü 2011] Ünlü, Saban. Cross-Plattform Entwicklung für Mobile-Endgeräte mit Flash. http://v4.nettrek.de/downloads/bt/CrossPlattform.pdf. Zuletzt geprüft am 29.08.2011.

[Widjaja01 2011] Widjaja, Simon. Hero, Burrito, Android. weave, 10 (2011, S. 79.)

[Widjaja02 2011] Widjaja, Simon. Multiscreen mit Flex mobile. weave, 13 (2011, S. 68.)

[Wroblewski 2010] Wroblewski, Luke. Touch Target Sizes. http://www.lukew.com/ff/entry.asp?1085. Zuletzt geprüft am 11.07.2011.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit zum Thema

„Untersuchung zur Verwendung von Adobe AIR als plattformübergreifende Laufzeitumgebung für mobile Applikationen anhand der prototypischen Umsetzung eines Taskmanagers."

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

Dresden, den 27. September 2011

_____

Silvia Graumann