

iPhone-Architektur und Programmierung (Juni 2010)

Abstrakt—Dieser Artikel gibt einen Überblick über die Architektur des iOS, dem Betriebssystem des iPhone, sowie die Softwareentwicklung dafür mithilfe des bereitgestellten SDK.

Index Terms—iPhone, Programming, Architektur

I. EINFÜHRUNG

Mit der Einführung des Apple App-Stores im Juni 2008 hat sich der Markt für Anwendungen auf Mobilgeräten grundlegend verändert. Durch die große Basis der Appstore-Fähigen Endgeräte und damit potentiellen Kunden, sowie der einfachen Vertriebung von Anwendungen über den Appstore wurde, angeregt von Erfolgsgeschichten wie zum Beispiel Flight Control¹, ein regelrechter Goldrausch der Entwickler ausgelöst. Das iOS ist das Standard-Betriebssystem der Apple-Produkte iPhone, iPod touch und iPad und basiert auf Mac OS X, dem von Apple entwickelten Betriebssystem für Macintosh-Computer.

II. DAS IPHONE

Mit dem ersten iPhone aus dem Jahre 2007 wurde eine neue Ära auf dem Smartphonemarkt eingeleitet. Begleitet mit einer ausschweifenden Berichterstattung in den Medien war es das erste Gerät, das mobiles Internet bereit für den Massenmarkt gemacht hat und zusammen mit der Eröffnung des Appstores im Jahre 2008 wurde ein Tor für eine komplett neue Art von Anwendungen für die Softwareentwickler aufgestoßen. Dabei bieten sich insbesondere durch das neue Bedienkonzept und eine Vielzahl von eingebauten Komponenten neue Möglichkeiten für die Entwickler.

Eine Besonderheit des iPhones ist das große, multitouchfähige Display und die Absenz von Knöpfen und Tasten zur Eingabe an dem Gerät. Das iPhone besitzt lediglich einen Knopf zum Ein-/Ausschalten, Stummschalten, zwei Knöpfe zur Lautstärkeregelung (seit 2. iPhone Generation) und den sogenannten „Home-Button“, über den Anwendungen beendet und zum Homescreen zurückgekehrt werden kann. Der Homescreen ist die Schaltzentrale des iPhones, von dem aus die Apps gestartet werden können.

Die Absenz physischer Eingabemethoden bedeutet

insbesondere, dass die Eingabe vom User an die verschiedenen Anwendungen hauptsächlich über den Touchscreen laufen muss, der bei Berührung so genannte *Events* absetzt, die dann von dem Programm verarbeitet werden können. Das iPhone unterstützt 5 gleichzeitige Berührungspunkte auf dem Touchscreen, das iPad sogar 11 verschiedene Punkte.

Zudem hat das iPhone mehrere Sensoren und Geräte eingebaut die von den Apps ausgelesen und angesteuert werden können. Das aktuelle iPhone 4 hat z.B. einen Beschleunigungssensor, ein Gyroscope, einen Annäherungssensor, einen digitalen Kompass und einen Umgebungslicht-Sensor. Zudem stehen eine rückseitige Kamera (für Einzelbilder und Video) und eine Kamera an der Vorderseite (für Videotelefonie), Mikrofon, Lautsprecher, Vibrationsgenerator, GPS und Unterstützung für Bluetooth, Wlan und GPRS/EDGE/UMTS zur Verfügung.

Durch diese Kombination von Features können so neue Anwendungen entwickelt werden, die so vorher nicht möglich waren und ein neues Usererlebnis bieten können, insbesondere in Bezug auf die Eingabemethoden und Portabilität des iPhones.



*Abbildung 1: Das iPhone 4,
Foto © Apple*

¹Flight Control wurde von 3 Mitarbeitern der Firma Firemint entwickelt und wurde bereits drei Monate nach Veröffentlichung 1 Million mal über den Appstore verkauft, nach einem Jahr über 2 Millionen mal. Quelle: <http://firemint.com/?p=707> (stand 01.07.2010)

Bei der Programmierung für das iPhone berücksichtigt werden muss aber natürlich auch die limitierte Rechengeschwindigkeit gegenüber eines Desktop-PCs oder

Laptops und eventuell langsame Netzwerkanbindung über das mobile Datennetz, genau wie der relative geringe Speicher, weshalb der Speicherverwaltung besondere Bedeutung beim Programmieren von Apps zukommt.

Neben den verschiedenen Versionen des iPhones gibt es auch den iPod touch und das iPad, die auch beide das iOS als Betriebssystem haben und ähnliche Features bieten. Fast alle Programme die für das iPhone geschrieben wurden laufen auch auf dem iPod Touch und dem iPad.

Inzwischen wurden über 50 Millionen iPhones, 35 Millionen iPod Touchs² und 3 Millionen iPads³ verkauft. Stand 7. Juni 2010 gibt über 225.000 Apps von Drittentwicklern im Appstore, die insgesamt über 5 Milliarden mal heruntergeladen wurden⁴.

III. ENTWICKLUNG DES iOS

Vorgestellt wurde das damals noch nicht weiter benannte Betriebssystem für das iPhone im Januar 2007, zusammen mit der Vorstellung des ersten iPhones. Die Namensgebung iPhone OS kam erst im März 2008, im Hinblick auf die Veröffentlichung der zweiten Version des Betriebssystems, die im Juli 2008 veröffentlicht wurde. Erst damit kam auch die Möglichkeit für Drittentwickler, eigene Programme für das iPhone OS zu schreiben, da die zweite Version des iPhone OS den Appstore einführte und das offiziellen SDK enthielt, das im März 2008, veröffentlicht wurde.

Mit der 3. Version des iPhone OS im Juni 2009 kamen neue Funktionen und Schnittstellen hinzu, die die Entwickler für ihre Apps nutzen konnten.

Zusammen mit der 4. Version wurde das Betriebssystem im Juni 2010 in iOS umbenannt und brachte weitere nutzbare APIs und Features mit sich, unter anderem die Möglichkeit des Multitaskings für Drittprogramme und die Apple-eigene Werbeplattform iAd.

Das iOS basiert auf dem Betriebssystem Mac OS X von Apple und bietet für Entwickler mehrere Frameworks in Objective C und C für die Entwicklung eigener Anwendungen. Im Folgenden sollen ein paar Besonderheiten des iOS hervorgehoben werden.

IV. DESIGN PATTERNS BEI DER IPHONE-ARCHITEKTUR

Bei der Programmierung von iPhone Anwendungen kommen verschiedene Design Patterns zum tragen, nach denen der Programmcode strukturiert wird. Das wichtigste dabei ist das Model-View-Controller-Pattern.

A. Model View Controller

Das Model-View-Controller-Pattern beschreibt die Trennung

von Code der für die Anzeige, Datenspeicherung und Geschäftslogik zuständig ist. Eine iPhone App besteht also normalerweise aus mehreren View-Objekten die das Layout das auf dem Display angezeigt wird vorgibt und auf die Eingaben des Users reagiert, sowie mehreren Controllern in denen Spezifiziert ist, was bei einer entsprechenden User-Interaktion geschehen soll und eventuell einem Datenmodell, was auch zur persistenten Speicherung von Daten genutzt werden kann. Damit werden Model, View und Controller voneinander getrennt, was einerseits der Übersichtlichkeit des Codes dient (da kein Code für die Darstellung mehr mit Geschäftslogik vermischt wird), und lässt sich zudem leichter anpassen, um zum Beispiel für iPhone und iPad die gleichen Controller aber unterschiedliche Views zu benutzen.

Viele Frameworks die das iPhone SDK mitliefert stellen bereits entsprechende Controller- und View-Klassen bereit, die in die eigene Anwendung eingebunden werden können um die entsprechenden Funktionalitäten zu realisieren.

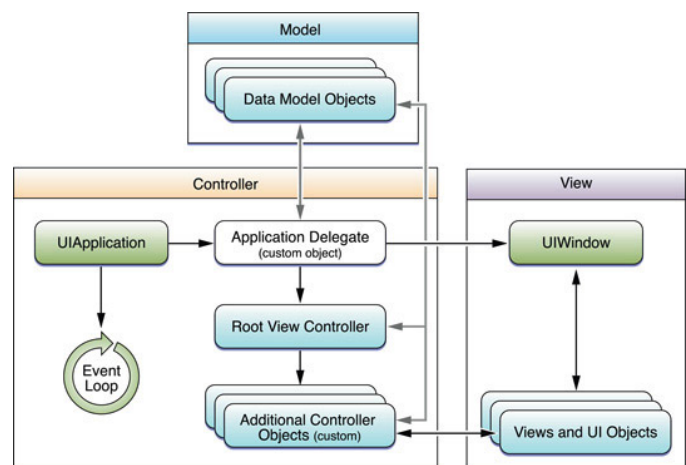


Abbildung 2: Anwendung des MVC-Entwurfsmusters im iOS, © developer.apple.com

B. Target Action

Für die Verbindung zwischen View und Controller kommt das Target-Action Pattern zu Tragen. Wenn der User mit dem View interagiert, werden sogenannte Events ausgelöst (zum Beispiel gibt es Events für das Berühren oder Loslassen des Bildschirms), und zu einem Event kann dann spezifiziert werden, welche Funktion in welcher Controller-Klasse aufgerufen werden soll.

Diese Funktionen im Controller, die von dem View aus aufgerufen werden, werden *Actions* genannt und dienen dazu, die Verbindung von dem View aus zu dem Controller herzustellen.

Die gegenteilige Richtung zu einer Action ist ein *Outlet*. Über Outlets kann vom Controller aus auf die verschiedenen Elemente im View zugegriffen werden, um so zum Beispiel den Inhalt eines Eingabefeldes auszulesen, oder den Text eines Labels zu verändern.

²Nach Steve Jobs bei einem Event zur Vorstellung des iPhone OS 4 am 08.04.2010

³Nach <http://www.apple.com/pr/library/2010/06/22ipad.html>

⁴Nach <http://www.apple.com/pr/library/2010/06/07iphone.html>

C. Subclassing

Ein eigentlich grundlegendes Prinzip der Objektorientierten Entwicklung ist Subclassing. Subclassing beschreibt, dass eigene Klassen von anderen Klassen abgeleitet werden können, und damit das Verhalten von diesen Klassen erben. So können Grundfunktionalitäten von anderen Klassen geerbt werden, diese dann aber noch überschrieben und erweitert werden.

Auch Subclassing wird wieder von vielen Frameworks verwendet, zum Beispiel erbt jeder View-Controller von der Klasse UIViewController, die bereits Grundfunktionalitäten für einen View bereitstellt.

D. Notifications

Auch eine Art des Observer-Entwurfsmusters kann in dem iOS eingesetzt werden. Dafür existieren sogenannte *Notifications* und das *Notification center*, das als Verteiler verstanden werden kann. Jedes Objekt kann sich beim Notification Center als Observer zu einer bestimmten Notification registrieren und jedes Objekt kann eine Notification an das Notification Center senden, die dann an alle registrierten Observer weitergeleitet wird. Somit können mehrere Objekte untereinander kommunizieren, ohne sich direkt kennen zu müssen. Einsatzzweck ist zum Beispiel das neue Anordnen der GUI-Elemente sobald eine Notification empfangen wurde, dass sich die Größe des zugehörigen Views verändert hat (zum Beispiel durch Drehen des iPhone).

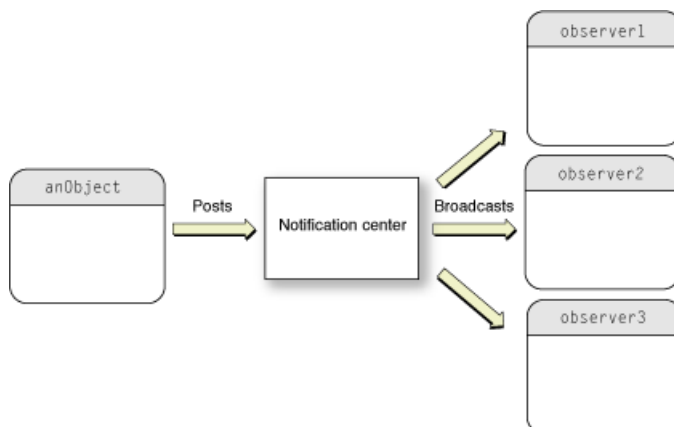


Abbildung 3: Der Einsatz des Notification Centers des iOS, © developer.apple.com

E. Delegation

Als letztes Beispiel für Design-Patterns bei der iPhone-Entwicklung sei noch Delegation genannt. Delegation erlaubt es, dass bestimmte Klassen die Verantwortung für einzelne Funktionen an andere Klassen abgeben können. Damit muss nicht immer, wenn etwas von der Standardklasse abweichen soll, eine Unterklasse davon erzeugt werden um die

entsprechende Funktion zu überschreiben, sondern Funktionen können einfach an andere Klassen delegiert werden, die dann deren Ausführung übernehmen. Dies kommt zum Beispiel bei dem Hauptcontroller einer App vor, der AppDelegate Klasse, die normalerweise die Funktion zum Anzeigen des Views nach dem Anwendungsstart an einen anderen Controller übergibt.

Die aufgerufene Funktion kann außerdem auch eine Rückgabe liefern die von Bedeutung ist. Beim Schließen eines Fensters zum Beispiel kann eine `windowShouldClose`-Nachricht an das delegate object gesendet werden. Dieses entscheidet dann, ob das Fenster geschlossen werden darf oder nicht (zum Beispiel weil eingegebene Daten noch nicht gespeichert wurden) und kann zum Beispiel auch ein Popup machen, ob das Fenster wirklich geschlossen werden soll.

V. GRUNDARCHITEKTUR EINER IPHONE APP

Ein großer Unterschied zwischen der Entwicklung von Desktop-Anwendungen und Apps für das iPhone ist das Prinzip der Windows und Views. An einem Desktop-Rechner ist man es gewohnt, dass selbst ein Programm mehrere Fenster offen hat, mit verschiedenen Elementen und Inhalten. Eine iPhone App hat nur ein Fenster, welches den kompletten Bildschirm ausfüllt.

Außerdem kann auf dem iPhone immer nur eine Anwendung gleichzeitig laufen (mit dem iOS 4 kam zwar eingeschränkte Unterstützung für Multitasking, echtes Multitasking existiert aber trotzdem nicht) und jede App läuft aus Sicherheitsgründen auch in einer eigenen Sandbox-Umgebung, in der ihre Einstellungen und Dateien gespeichert werden. Interaktionen zwischen zwei Apps können also nur über die von iOS spezifizierten Interfaces laufen.

Außerdem sind keine eigenen Plugins oder Frameworks erlaubt, andernfalls wird die App nicht für den Appstore freigegeben. Als Programmiersprache kann hauptsächlich Objective-C, aber auch C und C++ verwendet werden.

VI. DIE VERSCHIEDENEN EBENEN DES iOS

Das iOS ist in verschiedene Abstraktionsebenen unterteilt, die immer weiter von der Hardwareimplementierung abstrahieren und verschiedene Frameworks zur Verfügung stellen, die der Entwickler benutzen kann um häufig benötigte Funktionalitäten zu implementieren. Die vier Ebenen und ihre zugehörigen Frameworks werden im Folgenden näher erläutert.

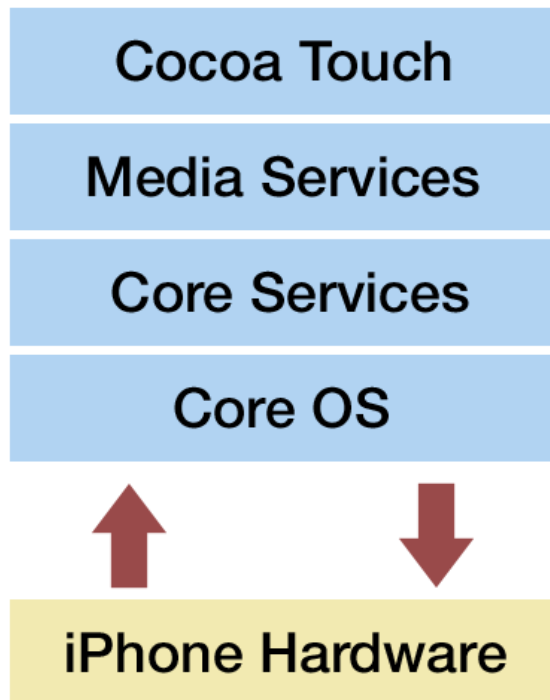


Abbildung 4: Einteilung des iOS in verschiedene Abstraktionsebenen

A. Cocoa Touch

Der Cocoa Touch-Layer ist die oberste Ebene mit dem höchsten Abstraktionsgrad und enthält die Frameworks, die am meisten von den Entwicklern genutzt werden. Dies sind die wichtigsten Frameworks und sie bilden die Grundlage einer iPhone App.

Viele Funktionalitäten, die auf dieser Ebene bereitgestellt werden, könnten auch durch die Implementierung von Frameworks aus den tieferen Ebenen erreicht werden, nach Möglichkeit sollte aber immer das Framework mit dem höchsten Abstraktionsgrad verwendet werden und nur wenn es nicht anders geht, auf die tieferen Ebenen zurückgegriffen werden. Zum einen können dadurch Fehler vermieden werden, die bei der komplizierteren Implementierung entstehen könnten. Zum Anderen beinhalten viele der im Cocoa Touch-Layer bereitgestellten Frameworks auch direkt die View-Elemente, legen also gleich das Layout fest, das der User beim Aufruf der Funktion angezeigt bekommt. Das dient der Übersichtlichkeit und bringt Konsistenz in das Design von Apps auf dem iPhone, wenn zum Beispiel jede App die Kontaktdaten aus dem Adressbuch auf die gleiche Weise darstellt.

Insbesondere enthält der Cocoa Touch Layer folgende Frameworks:

1) UIKit Framework

Das UIKit Framework ist das wichtigste Framework einer grafischen, Event-gesteuerten iPhone App. Es ist zuständig für

das komplette User Interface, also Formulare, Buttons, Texte, Fenster, Popups, Animationen, Web-Views ect. Zudem laufen alle Touch- und Gesten-Events über das UIKit. Darüber kann also gesteuert werden, was passieren soll, wenn ein bestimmter Bereich am Bildschirm berührt wird, oder eine Geste (z.B. Fingerstreich von links nach rechts, oder Zusammenführen zweier Finger) ausgeführt wird.

Weitere Features die über das UIKit Framework eingebaut werden können sind der Apple Push Notification Service, Unterstützung für Copy&Paste, Verknüpfung mit anderen Applikationen, PDF-Erstellung und viele mehr.

Auch der Support für Multitasking, der mit der iOS Version 4 eingeführt wurde, kann über das UIKit implementiert werden. Dabei handelt es sich allerdings nicht um echtes Multitasking, eine Anwendung kann lediglich Anfordern, nach der Beendigung noch eine bestimmte Zeit weiterlaufen zu dürfen, um noch eine wichtige Aufgabe zu erledigen (z.B. Fertigstellen eines Downloads), oder zu regelmäßigen Zeiten im Hintergrund laufen zu dürfen.

Außerdem kann über das UIKit noch Zugriff auf verschiedene Teile der Hardware genommen werden. Zum Beispiel können die Daten des Neigungssensors ausgelesen werden, oder mit der verbauten Kamera (soweit vorhanden) Fotos und Videos aufgenommen werden, oder der Batteriestatus und Informationen über ein angeschlossenes Headset oder das Gerät selber ausgelesen werden..

Das UIKit enthält insgesamt über 100 Klassen.

2) Message UI Framework

Mit dem Message UI Framework können E-Mail und SMS direkt aus der App heraus verschickt werden. Der Zusatz "UI" im Namen des Frameworks deutet schon an, dass dieses Framework sowohl Controller als auch Views enthält. Das bedeutet, dass beim Einbauen und Aufrufen der Funktion in die App, dem User bereits ein vorgefertigtes View angezeigt wird in der die Daten z.B. der E-Mail stehen, und der User sie dort überprüfen, ändern und absenden kann.

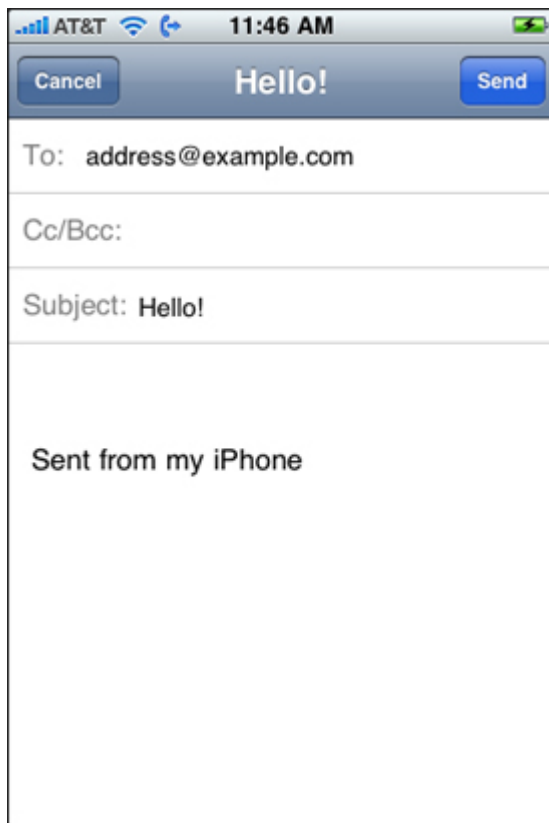


Abbildung 5: Beispiel für den vorgefertigten View zum E-Mail-Versand des Message UI Frameworks, © developer.apple.com

3) Map Kit Framework

Das Einbauen von Karten in die eigene App funktioniert über das Map Kit Framework. Es können damit Landkarten von Googlemaps direkt in der App angezeigt werden, inklusive Routenberechnung und Markierung bestimmter Standpunkte, oder auch die standalone Maps-App mit bestimmten Parametern gestartet werden.

4) iAd Framework

Das iAD Framework wurde mit der iOS Version 4 eingeführt und soll eine einfache und einheitliche Basis bilden für die Anzeige von Werbung in der App und insbesondere der Verarbeitung dieser, so dass also automatisch das Anzeigen oder ein Click auf die Werbung gespeichert wird und später eine entsprechende Auszahlung seitens Apple stattfindet.

5) Game Kit Framework

Mit der iPhone OS Version 3 wurde das Game Kit Framework eingeführt, mit dem Peer-to-Peer-Verbindungen über Bluetooth oder Netzwerk hergestellt werden können, sowie Voice-Chat zwischen zwei Geräten ermöglicht wird. Anstatt die Netzwerkfunktionalität über Netzwerkkomponenten aus den tieferen Ebenen des Betriebssystems zu implementieren, werden diese hier

abstrahiert und können so einfach verwendet werden. Und auch wenn die Bezeichnung des Frameworks „Game Kit“ ist, ist die Benutzung dieser Frameworks natürlich nicht nur auf Spiele-Apps beschränkt.

6) Event Kit UI Framework

Über das Event Kit kann Zugriff auf den auf dem iPhone befindlichen Kalender des Users genommen werden, Termine angezeigt, bearbeitet und hinzugefügt werden.

7) Address Book UI Framework

Das Address Book UI Framework bietet Zugriff auf das Adressbuch des Gerätes.

B. Media Services

Die zweite Abstraktionsebene im iOS ist der Media-Layer. Er bietet verschiedene High- und Lowlevel-Frameworks um Multimedia-Elemente, Audio, Video, Grafik und Animation in einer App zu verwenden. Einfache 2D-Grafiken, Vektorgrafiken und Animationen können ebenso dargestellt werden wie, verschiedene Audio-Formate. Ganze Videos mit Bild und Ton in mehreren Formaten können über das MediaPlayer Framework eingebunden und abgespielt werden. Für besonders hohe Performance und Anpassbarkeit stehen außerdem OpenGL ES (Open GL for Embedded Systems) und OpenAL zur Verfügung.

Insbesondere enthält der Media Layer folgende Frameworks:

1) Assets Library Framework

Das Assets Library Framework bietet einen Query-basierten Zugriff auf die Fotos und Videos auf dem iPhone. Außerdem können damit auch neue Fotos und Videos in dem Fotoalbum des Users gespeichert werden.

2) AV Foundation Framework

Das AV Foundation Framework dient dem Abspielen von Audio und Videodaten jedweder Länge. Es können mehrere Sounds gleichzeitig abgespielt und auch bearbeitet werden. Außerdem können damit auch Audio- und Videodaten von dem iPhone aufgenommen werden.

3) Core Audio

Mit Core Audio können Audiodaten erzeugt, aufgenommen, bearbeitet und abgespielt werden. Außerdem kann über Core Audio noch der Vibrationsgenerator gesteuert werden.

4) Core Graphics Framework

Das Core Graphics Framework ist die Schnittstelle zu der Quartz 2d drawing API, eine vektorbasierende Grafikingine aus Mac OS X.

5) Core Text Framework

Das Core Text Framework ist eine high Performance Text

Rendering Engine.

6) *Core Video Framework*

Unterstützung für die gepufferte Wiedergabe von Videodaten bringt das Core Video Framework.

7) *Image I/O Framework*

Mit dem Image I/O Framework können Bilddaten und Metadaten verschiedener Datentypen ausgelesen und gespeichert werden.

8) *Media Player Framework*

Über das Media Player Framework können auf einfache Weise einzelne Videos und Audiodateien abgespielt werden. Seit dem iPhone OS 3.0 kann darüber auch Zugriff auf die Playlist des Users genommen werden, um direkt in der App eine Liste der Lieder anzuzeigen und abspielen zu können.

9) *OpenAL Framework*

Das OpenAL Framework ist die Schnittstelle zu OpenAL.

10) *OpenGL ES Framework*

Das OpenGL ES Framework ist die Schnittstelle zu OpenGL ES.

11) *Quartz Core Framework*

Das Quartz Core Framework enthält die Schnittstellen zu den Core Animations, den Animationen und Effekten die zum Beispiel von dem UIKit bei der Anzeige der GUI verwendet wird.

C. *Core Services*

In der Core-Services-Ebene sind viele grundlegende Funktionen des iPhones enthalten, einige davon die auch schon über die Cocoa-Touch-Ebene und die Media Services erreichbar waren. Diese sind nun aber genauer konfigurierbar und auch von einem eventuell in einem Framework der höheren Ebenen direkt mitgelieferten View losgelöst.

Insbesondere werden folgende Frameworks zu den Core Services gezählt:

1) *Core Foundation Framework*

Das Core Foundation Framework bietet Grundlegende Klassen und Funktionen und ist deshalb auch standardmäßig in jedem Project eingebettet. Unter anderem bietet es verschiedene Datentypen für Collections (Arrays, Sets ect.), Klassen zur String-Verarbeitung, Zeit- und Datumsverarbeitung, Internationalisierung, Regulären Ausdrücken, Threading oder Caching.

2) *Foundation Framework*

Das Foundation Framework besteht im wesentlichen aus Objective-C Wrappern für die Features aus dem Core Foundation Framework

3) *Address Book Framework*

Ein direkter Zugriff auf das Adressbuch kann über das Address Book Framework erfolgen.

4) *CFNetwork Framework*

Über das CFNetwork Framework kann mit verschiedenen Netzwerkprotokollen gearbeitet werden. Unter anderem unterstützt es HTTP/HTTPS/FTP Aufrufe, Verschlüsselte Verbindungen über SSL oder TLS, BSD Sockets und Bonjour Services.

5) *Core Data Framework*

Das Core Data Framework wurde für die Datenschicht im Model-View-Controller-Design der Anwendungen entwickelt und sollte nach Möglichkeit verwendet werden, wenn lokal Daten gespeichert werden müssen. Es bietet unter anderem die Möglichkeit, Datenobjekte in einer lokalen SQLite-Datenbank zu speichern, ihre Inhalte zu validieren, Beziehungen zwischen Datenobjekten auf Konsistenz zu überprüfen und Daten im Hauptspeicher zu Gruppieren, Filtern und Sortieren.

6) *Core Location Framework*

Mit dem Core Location Framework kann die aktuelle Position (Längen- und Breitengrad) des Geräts bestimmt werden. Dabei wird das Gerät verwendet, das gerade zur Verfügung steht, also das GPS-Signal, oder Positionsangaben über das momentane Mobilfunknetz oder die WiFi-Station. Sofern vorhanden, kann hier auch auf den digitalen Kompass zugegriffen werden.

7) *Core Media Framework*

Das Core Media Framework bietet einen Low-Level Zugriff auf die Medientypen, die eigentlich durch das AV Foundation Framework wiedergegeben werden, für die Entwickler die mehr Kontrolle bei der Wiedergabe und Bearbeitung in ihrer Anwendung benötigen.

8) *Core Telephony Framework*

Über das Core Telephony Framework können Informationen über den aktuellen Netzbetreiber des iPhones ausgelesen werden. Dies kann zum Beispiel von Anwendungen des Netzbetreibers verwendet, die nur auf den Geräten der eigenen Kunden laufen sollen (zum Beispiel gibt es eine kostenlose Navigations-App von Navigon, die nur bei Kunden von T-Mobile funktionsfähig ist). Außerdem können noch Informationen über einen aktuell stattfindenden Anruf abgerufen werden.

9) *Event Kit Framework*

Ein direkter Zugriff auf den Kalender kann über das Event Kit Framework erfolgen.

10) *Quick Look Framework*

Mit dem Quick Look Framework kann eine Vorschau von Dokumenten angezeigt werden, die nicht direkt auf dem

iPhone verarbeitet werden können, wie zum Beispiel CSV-Dateien.

11) Store Kit Framework

Über das Store Kit Framework können so genannte In-App-Purchases abgewickelt werden. In-App-Purchases sind die Möglichkeit, Bezahlinhalte direkt in der App von dem User kaufen und via iTunes bezahlt zu lassen, um so die Zusatzinhalte freizuschalten (zum Beispiel neue Level bei einem Spiel, oder die aktuelle Ausgabe eines Magazins von einer Zeitschrift).

12) System Configuration Framework

Über das System Configuration Framework kann überprüft werden, ob eine WiFi oder Mobilfunknetz-Verbindung besteht und ein bestimmter Hostserver erreichbar ist.

D. Core OS

Core OS ist die niedrigste Ebene im iOS und enthält die Low-Level Features, auf die die meisten Frameworks der oberen Ebenen aufbauen. Normalerweise sollte nichts aus dieser Ebene direkt benötigt werden, sondern das benötigte Feature immer durch eine höhere Abstraktionsschicht realisiert werden können, in manchen Fällen kann der direkte Zugriff aber hilfreich sein. Frameworks in dieser Ebene sind:

1) Accelerate Framework

Das Accelerate Framework dient zur Berechnung Mathematischer Funktionen mit großen Zahlen und ist dabei für die jeweilig vorhandene Hardware optimiert. Egal also ob ein iPhone der ersten Generation, iPhone 4 oder iPad, alle mit unterschiedlichen Hardwarefeatures und Prozessoren, verwendet wird, wird durch das Accelerate Framework die Berechnung immer mit den gegebenen Mitteln und bestmöglicher Performance durchgeführt.

2) External Accessory Framework

Über das External Accessory Framework kann eine Verbindung zwischen externen Zubehör von Drittentwicklern und dem iPhone hergestellt werden, die über den 30-pin Dock Connector oder kabellos über Bluetooth angeschlossen sind. Somit muss sich die Softwareentwickeln für das iPhone also nicht nur auf neue Apps für den Appstore beschränken, sondern auch Hardwareerweiterungen in Verbindung mit Software können entwickelt werden.

3) Security Framework

Zusätzlich zu den eingebauten Sicherheitsfunktionen des iOS gibt es auch noch ein eigenes Security Framework, das die manuelle Verwendung von Verschlüsselung, Public und Private Keys ect. ermöglicht.

4) System

Die System-Ebene stellt die Grundlage des Betriebssystems

da, inklusive dem System Kernel, Gerätetreibern, Speicherverwaltung, Thread-Management, Dateisystem-Zugriff ect.

Aus Sicherheitsgründen haben Drittentwickler allerdings nur eingeschränkter Zugriff auf diese Komponenten, nur über die Library *LibSystem* kann auf ausgewählte Features des Systems zugegriffen werden.

Zusammenfassend bilden die Frameworks der verschiedenen Ebenen die Basis einer jeden iPhone App und decken die meisten benötigten Features ab. Die Benutzung eigener Frameworks ist nach dem Lizenzabkommen des Apple Developer Programs allerdings verboten, aus Sicherheits- und Kompatibilitätsgründen, da bei eigenen Frameworks nach einem Update des iOS nicht mehr gewährleistet ist, dass sie noch ordnungsgemäß funktionieren.

VII. DATENVERWALTUNG IM iOS

Bei vielen Datenintensiven Anwendungen müssen natürlich auch Daten über das Ausführen der Anwendung hinweg gespeichert werden, um auch beim nächsten Start noch verfügbar zu sein. Hierfür gibt es nun mehrere Möglichkeiten, die Daten persistent zu machen, und viele ähneln dabei den Möglichkeiten, die auch eine normale Desktop-Anwendung hat.

A. Ordner und Dateiesystem

Auch wenn das iPhone ein in sich geschlossenes System ist, so ist das Betriebssystem doch ähnlich zu anderen Computern heutzutage. Insbesondere gibt es auch im iOS ein Dateisystem das zum Speichern und Lesen von Dateien genutzt werden kann.

Da auf einem iPhone natürlich mehrere Apps installiert sein können, ist es aus Sicherheitsgründen so gelöst, dass jede App ausschließlich in ihrem eigenen Verzeichniss lesen und schreiben kann. Einer App stehen ein eigenes Arbeitsverzeichnis und ein eigenes temporäres Verzeichnis zur Verfügung, in dem Dateien gespeichert und auch Unterverzeichnisse beliebiger Tiefe erzeugt werden können. Auch Symbolische Links werden von iOS unterstützt.

B. Archiving

Da für das iOS normalerweise in Objective C objektorientiert programmiert wird, ist ein Anwendungsfall häufig auch nicht nur das Speichern einzelner Datensätze, sondern auch ganzer Objekte, um beim nächsten Start der Anwendung die Objekte wieder zur Verfügung zu haben. Anstatt nun die Daten aus dem Objekt manuell zu extrahieren und speichern, um anschließend beim lesen das Objekt wieder manuell zusammenzubauen, gibt es dafür im iOS das sogenannte *Archiving*, das aus anderen Programmiersprachen

auch als *Serialisierung* bekannt ist. Damit können ganze Objekte so encodiert werden, dass sie in eine Datei geschrieben werden und später auch einfach wieder ausgelesen und decodiert werden können.

C. SQLite

Wenn das vereinzelte Speichern von Daten oder Objekten in Dateien für bestimmte Anwendungen auch sinnvoll sein kann, ist für die meisten Datenintensiven Anwendungen das Speichern eben jener Daten in einer Datenbank doch der sinnvollste und effizienteste Weg. Das iOS bietet dafür die Möglichkeit, eine Datenbank mittels SQLite aufzusetzen.

SQLite ist ein Datenbanksystem das speziell für Embedded Systems entworfen wurde, und somit auch nicht auf einem Client-Server-Prinzip wie die meisten normalen Datenbanken basiert. Bei SQLite läuft kein Datenbankserver im Hintergrund, stattdessen kann direkt über die SQLite-Library im Programmcode auf die Daten zugegriffen werden. Dabei unterstützt SQLite die meisten Anfragen des SQL-92 Standards, inklusive Transaktionen, Subqueries, Views und Funktionen.

Eine iPhone App kann sich so mehrere Datenbanken anlegen die alle jeweils mehrere Tabellen mit unterschiedlichen Attributen enthalten können, und jede Datenbank wird dabei in einer eigenen Datei gespeichert.

Die SQLite-Library im iOS basiert auf C im Gegensatz zu den vorherigen zwei, Ansätzen, die objektorientiert in Objective C realisiert wurden.

D. Core Data

Während die SQLite-Library in iOS auf C basiert, wurde Core Data extra für die Bedürfnisse einer objektorientierten Anwendung in Objective C entwickelt. Core Data bietet die Möglichkeit, Daten objektorientiert zu speichern, wobei verschiedene unterliegende Speichermöglichkeiten angeboten werden. Standardmäßig wird eine SQLite-Datenbank zum Speichern verwendet, es ist aber auch möglich, die Daten als XML, im Binärformat oder nur im Hauptspeicher abzulegen. Gespeichert und Zugriffen wird außerdem nicht direkt über das entsprechende System, sondern über einen *Managed Object Context*, so dass unabhängig von der verwendeten Speicherart die gleichen Aufrufe gemacht werden können.

Entsprechend des Model-View-Controller-Entwurfsmusters einer iPhone-Anwendung gibt es in Core Data das so genannte *Managed Object Model*. In diesem Model wird festgelegt, welche Arten von Objekten mit welchen Attributen und Beziehungen untereinander gespeichert werden sollen, ähnlich dem Datenbankschema in einer relationalen Datenbank.

Ein einzelner Datensatz der über Core Data gespeichert oder gelesen wird, wird dabei durch ein sogenanntes *Managed Object* repräsentiert, das all die Attribute enthält, die vorher in dem *Managed Object Model* festgelegt wurden.

VIII. SPEICHERVERWALTUNG IM iOS

Mit dem begrenzten Arbeitsspeicher der iOS-Geräte (512MB iPhone 4, 256 MB iPhone 3GS, 128 MB iPhone 3G & original iPhone), wovon ein großer Teil noch von dem Betriebssystem selbst belegt wird, kommt der effizienten Verwaltung dieses Speichers natürlich besondere Bedeutung zu. Dies trifft insbesondere zu, da es im iOS *keine* Garbage Collection und *keine* Auslagerungsdatei gibt. Das bedeutet, dass für jedes Objekt, für das Speicher angefordert wurde, dieser Speicher auch wieder manuell freigegeben werden muss sobald das Objekt nicht mehr benötigt wird. Außerdem gibt es eben keinen zugesicherten, virtuellen Speicher der in einer Auslagerungsdatei gehalten werden könnte, sondern sobald der Speicher voll ist kann nichts mehr gespeichert werden, und die App stürzt im Zweifelsfall ab.

Um die Speicherverwaltung in den Griff zu bekommen gibt es deshalb besondere Techniken und Methoden, die in Abschnitt X B, Speicherverwaltung, näher erläutert werden sollen.

IX. DAS iPhone SDK

Das iPhone SDK ist für alle bei Apple als iPhone Entwickler registrierten User kostenlos erhältlich und enthält alle wichtigen Tools, um Apps für das iPhone zu schreiben und im Simulator zu testen. Um die Apps auf einem echten iPhone zu testen und in den Appstore zu stellen ist allerdings eine Mitgliedschaft im iPhone Developer Program für 99\$ im Jahr nötig (siehe Kapitel XI, iPhone Developer Program).

Das komplette SDK ist nur unter Mac OS X lauffähig, eine Entwicklungsumgebung für Windows oder Linux existiert nicht. Im Folgenden werden die wichtigsten Tools des SDK näher vorgestellt.

A. Xcode

Die Xcode IDE ist die zentrale Entwicklungsumgebung für Mac OS X und das iOS, vergleichbar mit Eclipse für Java.

Xcode enthält zu allererst einen Code Editor zum Schreiben des Quellcodes. Syntax Highlighting und Code Folding helfen den Quellcode zu strukturieren, durch Autovervollständigung und Quick Help wird der Entwickler beim Programmieren unterstützt und bekommt Informationen über die jeweiligen Funktionen direkt angezeigt. Auch die Codesyntax und statische Typen werden schon bei der Programmierung überprüft, wodurch viele Fehler bereits vor dem Kompilieren gefunden werden können.

Sollten es trotzdem Fehler in das Programm geschafft haben, enthält Xcode einen extensiven grafischen Debugger, mit dem sowohl im Simulator, als auch direkt auf einem angeschlossenen iPhone laufende Programme debugt werden können, mit dem setzen von Breakpoints und der Anzeige von Variablenwerten direkt im Editor.

```

tSentence
*sentence = @"";
*format;
unaryTable selectedRow ] != -1 ) {
tain *mountain = (Mountain *) [ [ self sortedMountains ] obj
mountain:climbedDate != nil ] {
format = [ % Mountain* mountain 0x223a90 Summary
name (fir) NSObject NSObject [...]
sentence = NSCFString* _name 0x218b20 Mount Wire
self date NSCFNumber* _height 0x222d10 2342
         NSCFDate* _climbedDate 0x2232f0 1979-07-2
format = NSLocalizedStringFromTable( @"undatedSentenceFormat
mountain's name (first parameter), and height (second para
sentence = [ NSString stringWithFormat:format, mountain.name

```

Abbildung 6: Der Debugger von Xcode in Aktion.

Xcode arbeitet projektbasiert, für jede Anwendung wird also ein Projekt erstellt, und Xcode liefert auch schon Templates für die verschiedenen Arten von Anwendungen mit (z.B. für auf Views basierende Apps, oder Apps die OpenGL ES zur Anzeige verwenden). Bestimmte gewünschte Features, wie zum Beispiel die Verwendung von Core Data zur Speicherung von Daten, können direkt bei der Erstellung eines neuen Projekts angewählt werden, um die dafür benötigten Komponenten gleich in das Template einzubinden.

Sobald das Programm getestet werden soll, kann es durch einen Klick auf den "Build and run"-Button direkt aus Xcode heraus kompiliert werden, wobei auch schon eventuelle Zertifikate eingebunden werden (siehe Kapitel XI A, Testen der eigenen App auf einem iPhone), und im Anschluss die Anwendung an den iPhone Simulator oder das iPhone übertragen und auch gleich gestartet wird.

Xcode enthält außerdem einen grafischen Entity Editor der genutzt werden kann, um zum Beispiel das in dem Abschnitt Core Data beschriebene Managed Object Model zu bearbeiten.

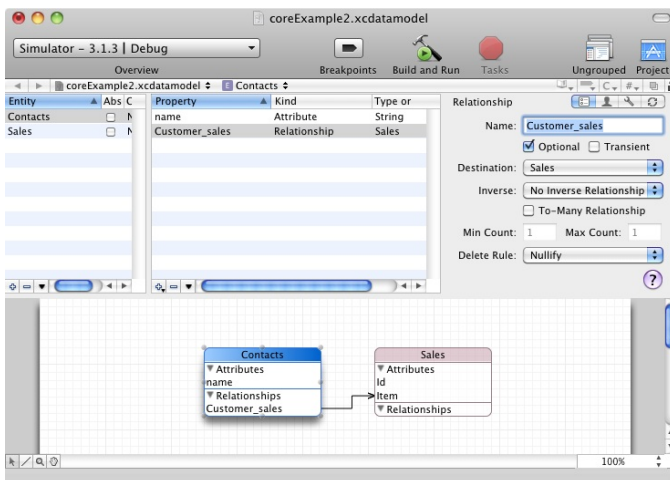


Abbildung 7: Der Entity Editor von Xcode

Außerdem unterstützt Xcode die Entwicklung im Team mit Hilfsmitteln wie zum Beispiel Subversion, und auch Möglichkeiten zum Refactoring des Quellcodes werden angeboten.

B. Interface Builder

Der Interface Builder wird verwendet, um die GUI einer Anwendung zu entwerfen. Er arbeitet nach dem Prinzip WYSIWYG (What you see is what you get), und bietet eine grafische Oberfläche, wo das entsprechende Fenster inklusive der verschiedenen Elemente (Eingabefelder, Buttons, Labels, Interviews ect.) angezeigt wird. Die Elemente können dort direkt verschoben und skaliert werden, wobei automatische Hilfslinien angezeigt werden, um eine symmetrische Anordnung der Elemente zu unterstützen.

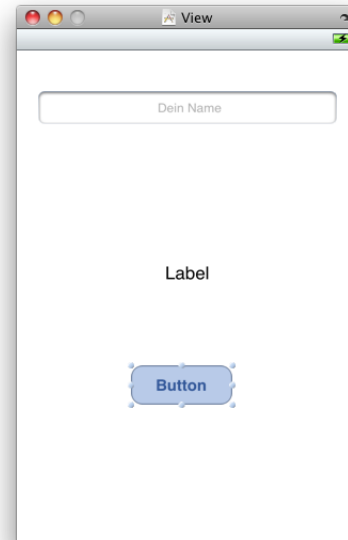


Abbildung 8: Anzeige des Views im Interface Builder

Neue Elemente können einfach aus einer Liste in den View hineingezogen und platziert werden, außerdem hat jedes Element auch eigene Eigenschaften und Parameter, die auch über den Interface Builder untersucht und angepasst werden können.

Die später stattfindende Verknüpfung zwischen den Elementen im View und Funktionen/Attributen in dem zugehörigen Controller erfolgt ebenfalls über den Interface Builder.

C. iPhone Simulator

Über den iPhone Simulator können sowohl ein iPhone 4G, als auch die vorherigen Versionen des iPhones und das iPad simuliert werden, um eine Anwendung darauf zu testen. Dabei können viele Aspekte des iPhones getestet werden:

- Bedienung per Touch mit einem Finger
- Pinch-Gesten (Zusammen- und Auseinanderführen zweier Finger)
- Drehen und Schütteln des iPhones
- Drücken des Buttons zum Sperren des iPhones

- Simulieren einer Warnung über zu wenig Hauptspeicher

Während der iPhone-Simulator also eine gute Methode ist, um die eigene Anwendung schnell und einfach zu Testen, fehlen natürlich auch viele Features die nur auf einem richtigen iPhone getestet werden können, wie zum Beispiel:

- Multitouch-Bedienung mit mehreren Fingern gleichzeitig
- Die Kamera des iPhones
- Mikrofon
- Accelerometer
- Gyroscope
- GPS (im Simulator werden immer statisch die Längen- und Breitengrade des Unternehmenssitzes von Apple in Cupertino, Kalifornien zurückgeliefert)

Außerdem ist die Performance in dem Simulator natürlich eine andere, wie auf dem jeweiligen Endgerät, weshalb jede App auch gründlich auf einem echten iPhone getestet werden sollte (siehe Kapitel XI A, Testen der eigenen App auf einem iPhone).

Um Screenshots von der laufenden Anwendung im Simulator zu machen empfiehlt sich das kostenlose Programm *iPhone-Simulator Cropper*.

D. Instruments

Ein wichtiges Tool, um die Performance und den Speicherverbrauch einer Anwendung zu untersuchen, was insbesondere für ein System mit begrenzten Ressourcen wie das iPhone von Bedeutung ist, ist Instruments.

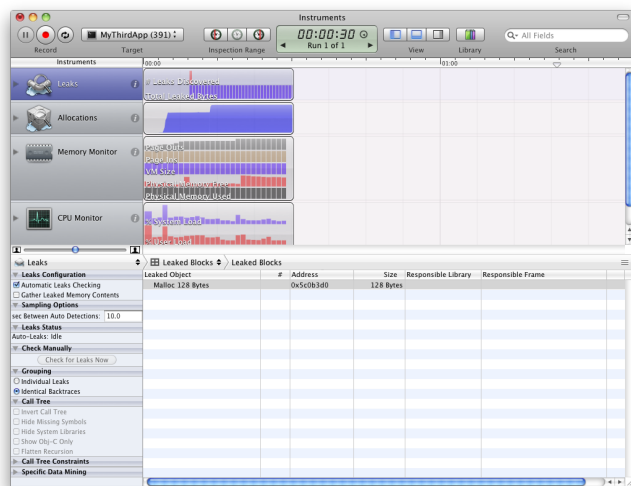


Abbildung 9: Hauptfenster von Instruments

Mit Instruments können ein oder mehrere Prozesse überwacht werden, und verschiedene Aspekte untersucht werden, wie zum Beispiel die CPU-Auslastung,

Speichernutzung, Anzahl der Zuweisungen, Netzwerkaktivität, Dateisystem-Aktivität oder Datenbankaktivität. Mit Instruments lassen sich Speicherlecks in der Anwendung aufspüren und kritische Stellen im Code identifizieren. Dabei bietet Instruments eine grafische Anzeige der verschiedenen untersuchten Aspekte, um Zusammenhänge gut erkennen zu können. Die Daten lassen sich während der Ausführung des Programms aufnehmen und für die spätere Analyse speichern, außerdem können auch die Interaktionen des Users mit der zu testenden Anwendung aufgezeichnet werden, um diese später erneut durchführen zu lassen und so die Auswirkung von Änderungen im Code auf die Performance testen zu können.

Andere Tools die für die Entwicklung und Analyse genutzt werden können, hier aber nur kurz genannt werden sollen, sind zum Beispiel Shark (genauerer Analysieren der Performance und Interaktion der App mit dem Betriebssystem, liefert Hinweise zur Optimierung) und der Quartz Composer (Grafisches Tool zum Entwerfen von Animationen für eine Cocoa-Anwendung).

E. Hilfen für den Einstieg

Nach der Installation von Xcode und dem ersten Start wird dem User gleich ein Fenster mit Hinweisen für den Einstieg in die iPhone-Programmierung mit Xcode präsentiert. So gibt es neben einer ausführlichen Dokumentation über das iOS und die enthaltenen Frameworks auch mehrere Artikel für den Einstieg, die den Umgang mit der Entwicklungsumgebung und die grundlegende Prinzipien bei der Entwicklung für iOS erklären. Außerdem gibt es ergänzend zu der Dokumentation auch viele Schritt-für-Schritt Tutorials, die den Umgang mit den Frameworks durch praktische Beispiele erläutern.

Ein besonderes Feature sind auch die Entwicklervideos die über iTunes kostenlos bezogen werden können, und die noch einmal in Bild und Ton einen Überblick über das iOS und die Softwareentwicklung dafür geben.

X. PROGRAMMIEREN FÜR DAS iOS

Das Programmieren für iOS unterscheidet sich in großen Teilen nicht zu dem für andere Plattformen, es gibt aber einige Besonderheiten, die beim Programmieren beachtet werden sollten und im Folgenden kurz vorgestellt werden.

A. Grundstruktur einer iPhone-App

Beim Anlegen eines neuen Projekts in Xcode kann bereits ausgewählt werden, welche Art von Anwendung man schreiben will. Zur Auswahl stehen:

- Window-based Application
- View-based Application
- Navigation-based Application

- Tab Bar Application
- Utility Application
- Open GL ES Application

Die Window-based Application ist dabei das grundlegendste der Templates, das nur ein Fenster und die AppDelegate-Klasse zum Aufrufen von anderen Klassen enthält. Alle anderen Möglichkeiten bauen auf dieser Window-based Application auf und bieten bereits ein oder mehrere Views und Controller.

Das Fenster (window) ist dabei das Grundelement einer jeden App und repräsentiert den Bildschirm, auf dem die verschiedenen Views der App angezeigt werden und auch die Eingaben per Touchscreen stattfinden, weshalb hier kurz auf die Verbindung von Windows, Views und View Elementen eingegangen werden soll.

Eine iPhone App besitzt normalerweise nur ein Window, und dieses Window steht ganz oben in der Hierarchie der GUI-Elemente. Es stellt die Fläche bereit, auf der alle Views und Subviews dargestellt werden. Ein Windows selber hat keinerlei angezeigte GUI-Elemente und der User kann auch nicht damit interagieren. In einem Window werden typischerweise ein oder mehrere Views angezeigt, die selbst wiederum Views und View-Elemente enthalten können, mit denen der User interagieren kann. Die verschiedenen View-Elemente können in folgende grobe Kategorien unterteilt werden:

- Container zum kompakten Anzeigen mehrere Datensätze, zum Beispiel in einer Tabelle oder zum scrollen.
- Kontrollelemente, wie zum Beispiel Buttons, Schalter, Eingabe- oder Auswahlfelder, mit denen der User interagieren soll.
- Elemente die nur der Anzeige dienen, wie Bilder oder Label, und mit denen nicht interagiert werden kann.
- Navigationselemente wie eine Tabbar am unterem Rand des Bildschirms, oder einer Navigationsleiste am oberen Rand.
- Text und Web-Views, zur Anzeige von einfachem Text oder Inhalten in HTML.
- Alert Views und Action Sheets, die als Popup zum Entgegennehmen von Benutzereingaben dienen.

Die verschiedenen Elemente können im View miteinander kombiniert und in beliebiger Tiefe verschachtelt werden, wodurch eine Hierarchie entsteht.

Die Eingabe des Users erfolgt dann durch so genannte *Events* die ausgelöst werden, wenn der User zum Beispiel eine Stelle des Bildschirms mit den Finger berührt, eine Geste ausführt oder den Finger wieder abhebt. Welche Aktionen ausgeführt werden bei einem Event, wird dabei auch durch die Hierarchie innerhalb der View-Elemente bestimmt.

Ein eingehender Event wird zuerst an das passende Element ganz unten in der View-Hierarchie gegeben, also zum Beispiel

an einen Button der gedrückt wurde. Kann das Element den Event nicht verarbeiten, hat also nicht spezifiziert, was bei jenem Event passieren soll, wird der Event an das nächst höhere Element weitergegeben, also zum Beispiel dem Subview indem sich der Button befindet. Dort wird dann eventuell der Event verarbeitet oder wiederum an das Elternelement weitergeleitet. Hat ein View einen eigenen Controller, wird der Event zusätzlich noch an den Controller geleitet und eventuell dort abgefangen, bevor er weitergeleitet wird.

Wenn ein Event nicht verarbeitet werden kann, wird es also immer zum nächst höheren Element weitergeleitet. Die View-Hierarchie bildet dort also eine art Zuständigkeits-Kette für die Events.

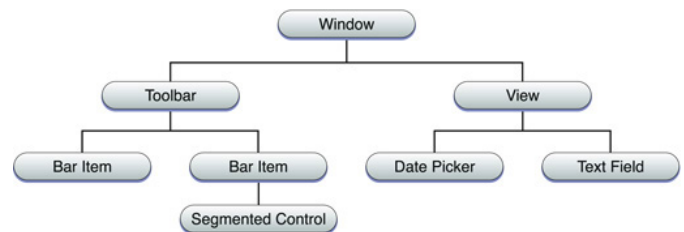
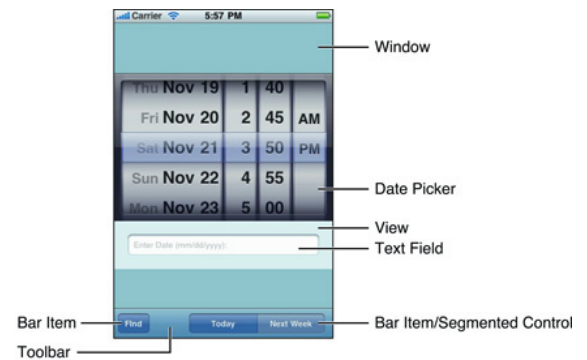


Abbildung 10: Beispiel eines Views und der zugehörigen Hierarchie, © developer.apple.com

Neben den Touch-Events gibt es auch noch Motion Events, wie zum Beispiel drehen oder schütteln des iPhones. Diese Events werden direkt von entsprechenden Funktionen in den Controllern verarbeitet.

Die meisten Anwendungen für das iPhone basieren auf dem *View-based Application-Template*, sind also Programme die auf dem Model-View-Controller Entwurfsmuster basieren. Während die Controller mit dem Source Editor geschrieben werden, und das Model mit dem Entity Editor entworfen werden kann, werden die Views typischerweise mit dem Interface Builder erstellt. Der wichtigste Dateityp für den Interface Builder sind dabei die .xib-Dateien, in denen ein View, mitsamt aller enthaltenen Elemente, gespeichert wird und in dem auch die Verbindung von View mit Controller stattfindet. Früher wurden diese Dateien vom Interface Builder nicht als .xib sondern .nib abgespeichert, weshalb sie auch

noch heute NIB-Files genannt werden.

Damit der Controller Zugriff auf die verschiedenen Elemente im View hat, also zum Beispiel auf den Text in einem Eingabefeld, muss für jedes Element auf das von dem zugegriffen werden soll ein *Outlet* erstellt werden. Ein Outlet ist im Prinzip eine Variable mit dem Schlüsselwort *IBOutlet* in der Controller-Klasse und wird über den Interface Builder mit dem zugehörigen Element im View verknüpft. Die Verknüpfung wird dabei direkt über den grafischen Editor des Interface Builders erstellt, indem das entsprechende Element im View markiert, und dann mit gedrückter Strg + Maustaste ein Pfeil zu dem File's Owner (dem Controller des Views) gezogen wird. Dort wird dann aus einem Popup die entsprechende Outlet-Variable ausgewählt.

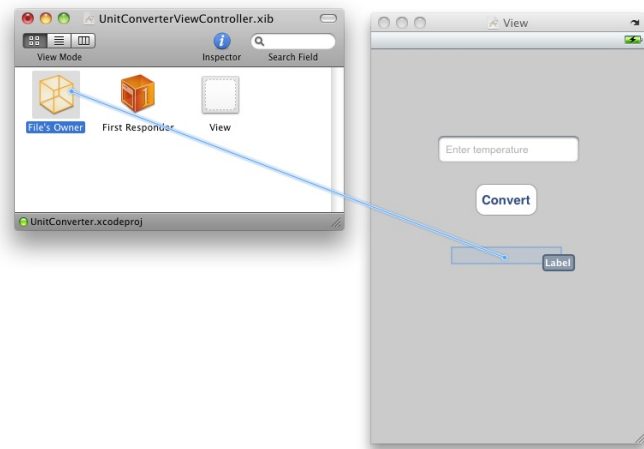


Abbildung 11: Verknüpfung eines View-Elements mit dem Controller über den Interface Builder

Über das Outlet in dem Controller können dann Attribute des Elements ausgelesen und auch geschrieben werden, um zum Beispiel die Beschriftung eines Labels zu ändern.

Das Gegenstück zu den Outlets sind die Actions. Actions sind Funktionen im Controller, die von dem View aus aufgerufen werden können. Dazu wird wiederum das Element im View ausgewählt (also zum Beispiel ein Button) und der Connections Inspector geöffnet. Dort sind die verschiedenen Events aufgelistet die auf dem Element ausgelöst werden können, zum Beispiel Touch Down oder Touch Up Inside. Von dem gewünschten Event als Auslöser kann dann wiederum mit Strg + Maustaste die Verbindung zu dem Controller gezogen werden, woraufhin die Action die ausgeführt werden soll ausgewählt werden kann.

IBAction und *IBOutlet* lauten die Schlüsselwörter, die in iOS für Actions und Outlet-Variablen verwendet werden müssen.

B. Speicherverwaltung

Wie bereits beschrieben muss bei der Programmierung für das iPhone besonders auf eine effiziente Speicherausnutzung geachtet werden. Da es in iOS keinen Garbage Collector wie

zum Beispiel in Java gibt, muss jeder manuell allokierte Speicher auch manuell wieder freigegeben werden, ansonsten entsteht ein Speicherleck das die App zum Absturz bringen kann.

Eine Grundregel ist dabei, dass jedes Objekt, das man selbst besitzt, auch von einem selbst wieder freigegeben werden muss, und zwar durch einen *release* oder *autorelease* Befehl. Man besitzt zum Beispiel alle Objekte, die man mit einer Funktion mit "alloc", "new" oder "copy" im Namen erstellt hat, oder für das man eine retain-Nachricht geschrieben hat (dies verhindert, dass ein Objekt aus dem Speicher entfernt wird, bis man es wieder freigegeben hat). Immer wenn ein Objekt also innerhalb einer Funktion mit einem solchen Befehl erstellt wurde, muss es spätestens am Ende der Funktion auch wieder released werden.

Generell wird dieses System der Speicherfreigabe *reference counting* genannt. Wenn ein Objekt neu erstellt wird, besitzt dies einen reference count von 1. Für jeden retain-Befehl der an das Objekt geschickt wird, erhöht sich der reference count um eins, für jeden release-Befehl wird er um eins erniedrigt. Sobald der reference count eines Objekts 0 erreicht hat, wird die dealloc-Methode aufgerufen und die Referenz aus dem Speicher entfernt.

Jede Klasse hat außerdem eine Funktion *dealloc*, die aufgerufen wird, wenn eine Instanz dieser Klasse vom Speicher entfernt wird. Dort können dann die einzelnen Klassenvariablen released werden, und danach die dealloc-Funktion der Oberklasse aufgerufen werden, um die Instanz selbst zu entfernen.

Wenn der Speicher knapp wird, sendet das iPhone außerdem eine entsprechende Meldung an die App, die in der Funktion *didReceiveMemoryWarning* im Controller bearbeitet werden kann. Diese Funktion sollte genutzt werden, um temporäre Daten zu löschen, zum Beispiel gecachte Grafiken, und somit wieder Speicher frei zu machen.

Aus dem gleichen Grund können sich Objekte auch als Observer der *UIApplicationDidReceiveMemoryWarningNotification* (siehe Kapitel IV D, Notifications) registrieren, um beim eingehen der entsprechenden Notification reagieren zu können.

C. Einbindung von C in Objective C

Eine Besonderheit von iOS ist, dass Programme neben Objective C auch C-Code enthalten können. Insbesondere liegen auch die bereitgestellten Frameworks unterschiedlich vor. Die meisten Frameworks aus den Core OS und Core-Services Ebenen sind zum Beispiel in C geschrieben, die der Media-Ebene sind gemischt C und Objective C, und die meisten Frameworks von Cocoa Touch sind in Objective C.

Um viele der Frameworks zu benutzen, müssen also C-Funktionen direkt aus dem Quellcode aufgerufen werden. Insbesondere ist dabei zu beachten, das die C-Funktionen mit den Basisdatentypen arbeiten und deshalb zum Beispiel Strings, die in Objective C normalerweise in die Klasse *NSString* gewrappert werden, vor der Übergabe an C-

Funktionen in einen einfachen String umgewandelt werden müssen, eben so wie eventuell die Rückgabe wieder in einen Objective-C Wrapper konvertiert werden muss, um sie mit anderen Objective-C Objekten zu verwenden. Deshalb ist es oft eine Überlegung wert, einen Wrapper in Objective-C für die entsprechenden C-Funktionen zu schreiben, der einem später die Arbeit des Umwandels erspart (oder eben direkt ein Framework aus den oberen Ebenen zu nehmen).

D. Speichern mittels Core Data

Core Data stellt die bevorzugte Art und Weise dar, Daten bei einer iPhone App zu speichern. Im Prinzip ist Core Data eine Art Wrapper für verschiedene Speichersysteme wie zum Beispiel SQLite in iOS. Mit Core Data ist kein Wissen über die SQL-Syntax mehr nötig, außerdem ist das Framework in Objective C geschrieben, anders als zum Beispiel SQLite, dessen Api in iOS in C geschrieben ist, und bietet so einen leichteren Zugang.

Core-Data stellt den Model-Teil einer Model-View-Controller-Anwendung dar, deshalb muss zuerst das Datenmodell definiert werden, in dem nachher die Daten gespeichert werden können. Dieses Modell befindet sich in der Datei "coreData.xcdatamodel", soweit beim Anlegen des Projektes auch die Verwendung von Core Data zur Speicherung angewählt wurde.

Mit einem Doppelklick auf die Modelldatei öffnet sich der Entity Editor von Xcode, in dem dann auf grafische Art und Weise das Modell erstellt werden kann. Dabei können verschiedene Entitäten erstellt werden, die mehrere Attribute verschiedenen Typs besitzen können, und auch Relationen unter den Entitäten (1-1, 1-n und n-n) können angelegt werden. Zu jedem der definierten Entities kann dann später ein so genanntes *Managed Object* erstellt werden, bei dem dann die Attribute gesetzt werden können und das anschließend gespeichert werden kann. Auch beim Auslesen der Daten werden Managed Objects der entsprechenden Entity zurückgeliefert. Es können also keine Instanzen von Klassen direkt gespeichert werden, wie beim Archiving, sondern alles läuft über die Managed Objects.

Das konkrete Speichern und Auslesen läuft dann über den *Managed Object Context*, der quasi eine Instanz der Datenbank darstellt, wodurch von der konkreten, unterliegenden Speicherstruktur abstrahiert wird. Auf diesem Managed Object Context können die Daten dann gelesen und gespeichert werden, wirklich persistent in den Speicher geschrieben werden. Änderungen aber erst, sobald ein Save-Befehl auf dem Managed Object Context ausgeführt wird (also ähnlich zu dem commit bei einer Transaktionsbasierten Datenbank).

E. Kommunikation mit anderen Apps

Eine direkte Kommunikation zwischen verschiedenen Apps ist im iOS nicht möglich, da immer nur eine App gleichzeitig laufen kann und Apps auch immer nur in ihrer eigenen

Sandbox-Umgebung laufen und nur dort Dateien verändern können. Um trotzdem auf einem einfachen Level mit anderen Apps zu interagieren, können URL-Schemas benutzt werden um andere Apps zu starten und direkt Parameter zu übergeben. Über das Öffnen der URL "myapp://action?param=value" wird zum Beispiel die App, die sich zu dem URL-Schema "myapp" registriert hat, geöffnet und bekommt die Parameter in der Funktion `didFinishLaunchingWithOptions` aufgerufen, die immer nach dem Start einer App aufgerufen wird.

Von Apple standardmäßig eingebaute URL-Schemas sind zum Beispiel `http` (öffnet Safari), `mailto` (Mail-App), `tel` (Telefon-App) und `sms` (Nachrichten-App). HTTP-URLs die auf YouTube, Googlemaps oder einer iTunes zeigen, werden auch automatisch abgefangen und starten anstatt den Safari-Browser die entsprechende App.

XI. IPHONE DEVELOPER PROGRAM

Während die Entwicklungstools des SDK grundsätzlich kostenlos sind und von allen registrierten Entwicklern heruntergeladen werden können, fällt für das Eintragen in das iPhone Developer Program von Apple eine Gebühr von 99\$ pro Jahr an. Dafür hat man dann nicht mehr nur die Möglichkeit, die eigene App im Simulator zu testen, sondern kann sie auch auf ein richtiges iPhone aufspielen um sie dort zu testen. Außerdem ist die Mitgliedschaft im Developer Program Voraussetzung dafür, die eigene App in den Appstore von Apple zu bekommen und darüber zu vertreiben.

Für eine Organisation mit mehreren Entwicklern muss nur einmalig ein Developer Account angelegt werden, die Mitarbeiter können daraufhin kostenlos eingeladen und in das Team aufgenommen werden, um Zugriff auf den Entwicklerbereich zu bekommen.

A. Testen der eigenen App auf einem iPhone

Im Gegensatz zum einfachen Testen der App auf dem iPhone-Simulator gestaltet sich insbesondere die erste Einrichtung zum Testen auf einem richtigen iPhone komplizierter. Jede App, die auf einem iPhone installiert ist, muss vorher mit einem iPhone Development Zertifikat zertifiziert worden sein, auch eigene Apps die nur zum Testen auf das iPhone installiert werden sollen.

Um die eigene App mit einem solchen Zertifikat zu versehen, muss zuerst über das Mac OS X Tool *Keychain Access* eine Zertifikatsanfrage mit den eigenen Daten erstellt werden. Diese Anfrage muss dann im Mitgliederbereich des iPhone Developer Programs, im sogenannten *Provisioning Portal*, hochgeladen werden. Dort wird das Zertifikat von dem Admin des Development Accounts bestätigt werden (wenn man selbst der Admin ist kann man sich das Zertifikat selbst bestätigen) und kann daraufhin heruntergeladen werden. Mit einem Doppelklick auf das heruntergeladene Zertifikat wird dieses dann in Keychain Access geladen und kann dort installiert werden.

Als nächstes müssen die Geräte ausgewählt werden, die für die Entwicklung benutzt werden und auf welche dann die Apps installiert werden können. Dazu hat jedes Gerät eine sogenannte UDID (Unique Device Identifier), einen 40-stelligen Code der das Gerät eindeutig identifiziert. Dieser Code kann über Xcode oder iTunes von dem angeschlossenen Gerät ausgelesen werden und muss dann wiederum von dem Team Admin im Provisioning Portal eingetragen werden.

Außerdem benötigt auch noch jede App die installiert werden soll eine App ID. Diese wird ebenfalls im Provisioning Portal erstellt, dort wird der Name der Anwendung angegeben und die App ID selbst, welche aus einem Prefix, der Bundle Seed ID, und einem Suffix, dem Bundle Identifier besteht. Während die Bundle Seed ID automatisch generiert wird, kann der Bundle Identifier frei angegeben werden. Empfohlen wird dazu den Domainnamen der Organisation in umgedrehter Reihenfolge zu nehmen, und den Namen der Anwendung anzuhängen, also zum Beispiel "de.uni-marburg.stundenplan".

Falls man eine ganze Gruppe von zusammenhängenden Apps schreiben will, kann der Bundle Identifier auch mit einem Wildcard-Zeichen versehen werden, also zum Beispiel "de.uni-marburg.*". Apps mit einem solchen Wild-Card-Identifier in der App ID können allerdings nicht in Verbindung mit dem Apple Push Notification Service oder In App Purchases verwendet werden.

Nun kann ein *Provisioning Profile* erstellt werden, indem die bisher angelegten Daten miteinander verknüpft werden. Dort wird festgelegt, welche Entwickler (anhand ihrer Development Zertifikate), welche App ID auf welchen Geräten (anhand der UDID) testen dürfen. Nachdem das Profil erstellt wurde, kann es mit Xcode auf ein Gerät installiert werden woraufhin das Gerät zum Testen dieser App ID freigegeben ist.

Schließlich muss die zu testende App noch in Xcode mit der zugehörigen App ID verknüpft werden und anschließend kann dann entweder das angeschlossene Gerät in Xcode ausgewählt werden und anstatt wie bisher die App in dem Simulator zu starten, kann sie nun direkt auf das iPhone installiert und gestartet werden sobald der "Build and run"-Button in Xcode gedrückt wurde, oder die kompilierte App wird via Ad Hoc Distribution an die Tester verteilt, d.h. sie bekommen die kompilierte App z.B. per E-Mail zugeschickt und können diese dann über iTunes auf das iPhone installieren.

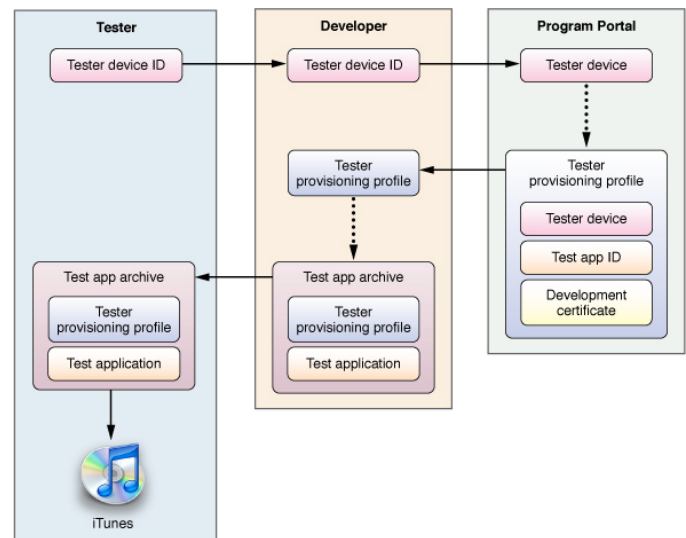


Abbildung 12: Testen einer App auf dem iPhone, © developer.apple.com

B. Einstellen der App in den Appstore

Um eine App in den Appstore von Apple zu stellen, und damit potentiellen Kunden zugänglich zu machen, wird ein anderes Profil als das Development Provisioning Profile benötigt, nämlich das sogenannte *App Store Distribution Provisioning Profile*. Wie beim Development Provisioning Profile muss dazu erst ein Distribution Zertifikat erstellt werden, woraufhin das Provisioning Profil erstellt, heruntergeladen und installiert werden kann. Vor dem Kompilieren der App in Xcode muss dann dieses Profil ausgewählt und der App Identifier der App eingegeben werden (im Identifier ist nun aber kein Wildcard-Zeichen mehr erlaubt, stattdessen muss nun der Name der Anwendung angegeben werden).

Nach dem Auswählen von "Distribution" als aktive Konfiguration in Xcode, kann das Projekt nun kompiliert werden und eine .app-Datei, die die Anwendung enthält, wird erstellt. Über iTunes Connect, die Webanwendung zum Verwalten von Apps, kann die App dann letztendlich zu Apple hochgeladen werden, wo sie dann überprüft und eventuell für den Appstore freigegeben wird.

Der Vorgang des Reviews einer jeden Anwendung und jedes Updates dient laut Apple dem Schutz der Privatsphäre des Users (wir haben ja bereits gesehen, dass eine App über die verschiedenen Frameworks Zugriff auf viele private Daten wie die Kontaktliste, Kalender oder Fotos hat), dem Schutz von Kindern vor unangemessenen Inhalten und dem Sicherstellen einer guten Usererfahrung auf iPhone, iPod touch und iPad. Deshalb wird jede Anwendung auf ihre Stabilität und Robustheit auf allen verschiedenen iOS-Geräten geprüft, und auch ob sie den Designrichtlinien für Apps auf dem iPhone/iPad entsprechen und nicht gegen das Lizenzabkommen des Apple Developer Programs verstößt. Sollte eine App nicht zugelassen werden, bekommt der

Entwickler dies in einer E-Mail mitgeteilt, in der auch der Grund für die Ablehnung und in den meisten Fällen direkt Hinweise zum Beheben des Problems enthalten sind.

Bei dem Vertrieb der eigenen App über den Appstore gehen 30% des Erlöses direkt an Apple, dafür sind dann alle entstehenden Kosten der Vertriebsplattform inbegriffen. Für kostenlose Apps werden überhaupt keine Gebühren verlangt.

Laut Apple werden 85% der eingereichten neuen Apps und 95% der Updates innerhalb von einer Woche freigegeben⁵.

LITERATURVERZEICHNIS

- [1] Apple, iPhone Developer Docuamenton [Online]. Available: <http://developer.apple.com/iphone> (stand 01.07.2010)
- [2] Techotopia, iPhone App Development Essentials [Online]. Available: http://www.techotopia.com/index.php/IPhone_App_Development_Essentials (stand 01.07.2010)

⁵Quelle: <https://developer.apple.com/iphone/appstore/approval.html> (Stand 01.07.2010)