Hochschule Fulda
University of Applied Sciences

Bachelor Thesis

Analyse von Touch Eingaben und Gesten zur Optimierung der Interaktionszeit anhand einer Android App

Martin Schmitt*

2. Dezember 2011

Eingereicht bei Prof. Dr. Hans-Helmut Paul

*624822, Martin.Schmitt@informatik.hs-fulda.de

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADT: Android Development Tools

API: Application Programming Interface

DDMS: Dalvik Debug Monitor Server

SDK: Software Development Kit

UI: User Interface

Abbildungsverzeichnis

1 Abstract

2 Einleitung

3 Motivation

Mobile Endgeräte sind auf dem Vormarsch, Handys werden fast aussschließlich als Smartphones produziert und Tables schmälern den Umsatz von PC und Laptopherstellern. Dabei erobern sie Bereiche in denen herkömmliche Computer bisher kaum Fuß fassen konnten. Durch die erhöhte Portabilität und eine Vielzahl intuitiver Eingabemöglichkeiten wie Akustik, Gestik und Position lassen sich mobile Endgeräte in Situationen nutzen in denen herkömmliche Geräte wie PCs, Note- und Netbooks nur unzureichend geeignet waren. So listet zum Zeitpunkt der Thesis der Android Market über 1000 verschiedene Apps zum Thema Notizen. Offensichtlich ist das Smartphone als multimediales Allzweckgerät gut geeignet um kurze Notizen zu speichern. Um sich gegen etablierte Medien durchzusetzen benötigen Sie jedoch neue intuitive und situationsangepasste Eingabemethoden.

Die Nutzung von Touchscreens an Stelle von Maus und Tastatur ermöglicht ein neues Interaktionsspektrum. Musste man früher die Tastatur als zusätzlichen Interpreter nutzen Informationen, die als Text vorliegen, in ein maschinell verarbeitbares Format zu bringen, so kann mit der Erfassung von Berührung und Bewegung ein wesentlich intuitiveres Medium zur Informationsübertragung genutzt werden. Durch zwischenmenschliche Interaktion ist der Benutzer mit der Nutzung von Gesten und Berührung als Informationsträger vertraut.

Entwickler müssen sich auf diese Situation einstellen und Bedienoberflächen entwickeln die sich an die aktuellen Begebenheiten anpassen, um den Nutzer die Möglichkeit zu geben die technischen Eingabevarianten der Geräte effektiv zu nutzen. Wo die Bedienung mit Maus und Tastatur einen künstlichen Umweg darstellt müssen die Entwickler und Software Designer nun umdenken. Herkömmliche Menü und Bildschirmaufteilungen sind oftmals überladen oder unpassed da die Software mit den unterschiedlichsten Auflösungen und variierenden Display-Formaten der mobilen Endgeräte umgehen können muss.

Anstelle eines herkömmlichen Menüs soll deshalb ein Kontext Menü implementiert werden, das sich bei Berührung eines Bildschirm-Elements aktiviert und durch eine Geste bestehend aus Richtung und Entfernung gesteuert wird. Anschließend wird dieses Menu mit 2 herkömmlichen Menüs verglichen um die Akzeptanz, Intuivität und Interaktionszeit zu testen.

Für den Test der Menüs wird ein Klassenbuch App Prototyp der Firma CAS Software AG verwendet, in deren Rahmen diese Bachelor Arbeit erstellt wurde. Die CAS Software AG ist ein innovatives Unternehmen im Bereich xRM Software mit einer breiten modularen Produktpalette an xRM Software. Zu den Hauptforschungsgebieten zählen neben den Web-Andwendungen mit Ajax und HTML5 die Entwicklung von mobilen Apps.

4 Theorie und Grundlagen

4.1 Grundlagen Gestik

Gesten können laut Cadoz in drei unterschiedliche Rollen aufgeteilt werden, die Semiotische, die Ergotische und die Epistemische Funktion. Obwohl sich viele weitere Unterteilungen vornehmen lassen und die drei genannten sich noch weiter spezifizieren lassen wird im Rahmen dieser Arbeit nur Typisierung einer Geste anhand ihrer grundsätzlichen Funktion behandelt.

Die Semiotische Funktion dient der Übertragung von Informationen durch Zeichen. Semiotik bezeichnet die Wissenschaft von Zeichen. Dabei werden Gesten als Symbole für Informationen genutzt. Diese sind jedoch nicht festgelegt und können sich zum Beispiel in den unterschiedlichen Kulturkreisen unterscheiden. Als typischen Beispiel dienen Handbewegungen wie z.B. das Winken zum Abschied oder der ausgestreckte Zeigefinger der auf etwas hinweist.

Die Ergotische Funktion bezieht sich auf die Verrichtung von Arbeit. Mittels einer Geste wird die Umwelt beeinflusst und verändert. Gegenstände können erschaffen und manipuliert werden. Dazu zählt zum Beispiel das Wischen von Staub oder das Formen von Objekten aus Ton. Gesten die eine ergotische Funktion erfüllen sind nicht eindeutig, da eine Geste z.B. mehrmals in Unterschiedlichen Zusammenhang gebraucht werden kann.

Die Epistemische Funktion bestimmt die Wahrnehmung unserer Umwelt durch Gesten. Mit Hilfe des Tastsinn können wir die taktilen und haptischen Eigenschaften von Objekten erfassen. So können wir Materialarten unterscheiden und Formen bestimmen.Dies kann genutzt werden um zum Beispiel Bedienelemente wie die Knöpfe einer Fernbedienung zu erkennen. [5]

4.1.1 Definition Geste

Eine Geste bezeichnet „jede bewusste oder unbewusste Körperbewegung, außer den Vokalisierungsbewegungen, durch die wir entweder mit uns selbst oder mit anderen kommunizieren". [3]
Zur Gestik zählt nach der Definition von Hayes neben der Bewegung von Armen und Beinen, auch die Mimik und die Körperhaltung. Da wir beim Touchscreen aber auf zweidimensionale Gesten mit Hilfe von Fingern beschränkt sind können wir nur ein sehr kleines Spektrum an Gesten erfassen. Durch die Ausnutzung moderner Multitouch Displays lässt sich jedoch die potentielle Anzahl an Gesten stark erhöhen. Um die Gesten digital zu verarbeiten erfassen wir eine Geste als eine Reihe von Vektoren in einem zweidimensionalen Koordinationsystem, wobei die Achsen durch die Höhe und Breite des Touchscreens definiert sind. Eine Geste besteht also aus einer oder im Falle von Multitouch aus mehreren Berührungen. Jede Berührung besteht aus einem Startpunkt, mehreren verketteten Vektoren und einem Endpunkt. Mit Hilfe dieser Information kann man eine Geste als zweidimensionales Symbol erfassen und ihr einen Informationsgehalt zuweisen. Deshalb sind diese Gesten in den Bereich der Semiotischen Funktion einzuordnen.

4.1.2 Bekannte Gesten

Betrachtet man allgemein gebräuchliche Touchscreen-Gesten so fällt auf das es sich dabei in den allermeisten Fällen um Abstraktionen von Interaktionen oder Kommunikation mit Objekten und Menschen aus der realen Welt handelt. Dabei werden die Gesten skaliert und auf die Beschreibung mittels Finger angepasst, um die Intuivität und somit die Akzeptanz beim Nutzer zu erhöhen. [1] Aufgrund der Fülle an Gesten wird hier nur ein kleiner Teil als Beispiel angeführt für Gesten die heute als selbstverständlich angesehen werden.

Pinch

Bei der sogenannten *Pinch* Geste, bewegen sich 2 Finger [1] aufeinander zu. Durch diese Geste signalisieren wir eine Reduzierung z.B. einer Menge oder einer Strecke. Überträgt man dieses Konzept auf Objekte des User Interface so lässt sich damit die Größe des berührten Objekts reduzieren.

[1] Vorrangig Daumen und Zeigefinger

Abbildung 4.1: Pinch (Entnommen aus [9])

Spread

Führt man die *Pinch* Bewegung in gegengesetzter Richtung aus und spreizt die Finger so wird das Objekt vergrößert. Diese Geste wird als *Spread* bezeichnet.

Abbildung 4.2: Spread (Entnommen aus [9])

Tap

Eine einfache, kurze Berührung des Touchscreens bei der sich die Position der Berührung nicht verändert. Diese Geste wird benutzt um ein Objekt auszuwählen oder zu eine Aktion auszulösen.

Abbildung 4.3: Tap (Entnommen aus [9])

Double Tap

Während der *Tap* aus der realen Welt abgeleitet werden kann übersetzt der *Double Tap* den Doppelklick einer Maus auf den Touch Screen. Da der Doppelklick keine natürlich vorkommende Geste ist, ist diese Geste weniger intuitiv als Beispielsweise der *Tap*.

Abbildung 4.4: Double Tap (Entnommen aus [9])

Drag

Bei der *Drag* Geste wird ein Objekt auf dem Touchscreen berührt und dann ohne die Berührung zu beenden eine Bewegung durchgeführt. Durch das Beenden der Berührung wird die Geste abgeschlossen und das berührte Objekt an die Endposition der Berührung verschoben. Dies wird benutzt um die Positionierung von Objekten zu ändern und bildet das Greifen und Ablegen von Objekten ab.

Abbildung 4.5: Drag (Entnommen aus [9])

Swing / Flick

Der *Swing* oder *Flick* ist eine schnelle Wischbewegung die zur Navigation innerhalb einer List von Objekten oder Zuständen dient. So kann Sie genutzt werden um eine Bildschirmseite umzublättern oder die Position in einer Liste schnell zu verändern. Ein *Swing* kann mit einem oder mehreren Fingern durchgeführt werden, es ist eine der wenigen Gesten bei der auch die Geschwindigkeit der Ausführung, neben der zurückgelegten Strecke und der Ausrichtung im Koordinatensystem, eine Rolle spielt.

Abbildung 4.6: Swing (Entnommen aus [9])

Rotate

Bei der *Rotation* wird der Bildschirm mit zwei Fingern berührt und diese starten eine Kreisbewegung in gleicher Richtung. Durch die kreisförmige Bewegung wird eine Drehung impliziert die auf ein berührtes Objekt auf dem Touchscreen angewandt wird. Zwar wäre es intuiver und weniger abstrakt eine solche Bewegung nur mit einer Berührung des Bildschirms durchzuführen, durch die zusätzliche Berührung lässt sich jedoch die Erkennungsrate der Geste steigern.

Abbildung 4.7: Rotate (Entnommen aus [9])

4.2 Tablet Technologie

Tablets oder Tablet-Computers entwickelten sich aus der Slate Bauweise der Tablet PCs. Dies bezeichnet Rechner die ohne Tastatur aber dafür mit einem Stift als Eingabeinstrument ausgestattet sind. Im Unterschied zum Tablet-Computer besitzen Tablet-PCs ein vollwertiges Betriebssystem wie es vom Desktop PC oder Notebook bekannt ist, während ein Tablet Computer nur in einem engen Rahmen mit eigens dafür angepassten Betriebssystem und Programmen, sogennanten Apps, auskommt. Während die Tablet-PCs ein Nischendasein führen wurden Tablet-Computer mit der Einführung des Apple IPad populär. [4]

Durch den hohen Marktanteil an Apple Geräten wurden die von Apple eingeführten Gesten zum Standard für Tablet-Computer. So asoziieren die meisten Personen die schon einmal einen Touch-Screen bedient haben, das Zusammenführen von 2 Fingern als Zoom oder das Wischen mit einem Finger auf vertikaler oder horizontaler Ebene als Swipe. Android konnte jedoch seinen Marktanteil binnen eines Jahres von 2,9% auf 30% steigern.[8]

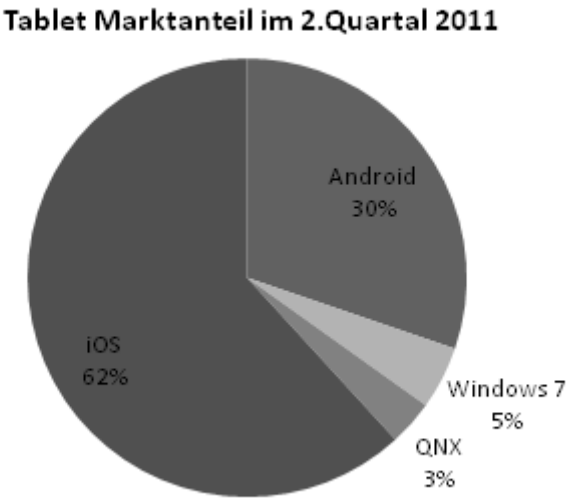

Abbildung 4.8: Android Marktanteil (Entnommen aus [8])

4.3 Grundlagen Android

4.3.1 Android Entstehung und Entwicklung

> Android is the first truly open and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications – all of the software to run a mobile phone, but without the proprietary obstacles that have hindered mobile innovation [7]

2005 kaufte Google die Firma Android die ein gleichnamiges Betriebssystem für mobile Endgeräte entwickelte. Das auf einem Linux-Kernel basierende Betriebssystem ist heute das am weitesten verbreitete Betriebssystem für mobile Endgeräte.Der größte Teil von Android ist als Open Source unter der Apache Lizenz veröffentlicht. (Wikipedia) Android wurde speziell für den Einsatz auf Smartphones und Tablets angepasst wobei die Version bis 2.3.7 für Smartphones optimiert waren und die Version ab 3.0 bis 3.2.1 für Tablets. Diese Aufspaltung wurde mit der Einführung von Ice Cream Sandwich, Version 4.0 wieder aufgehoben. Da mit Ice Cream Sandwich ein überarbeitetes UI eingeführt wurde, das für Tablets und Smartphones gleichermaßen geeignet ist.

4.3.2 Behandlung von Touch Events und Gesten in Android

Jede Android GUI ist hierarchisch gegliedert. An oberster Stelle steht immer ein *View* Objekt. Das *View* Objekt kann weitere Views oder deren Subklassen, sogenannte *Widgets*, enthalten. Die Referenz auf die Wurzel dieser Baum Hierarchie wird der Activity zugewiesen. Wird eine Eingabe erkannt startet die *Activity* im Vordergrund eine dispatch Methode. Handelt es sich dabei um eine Berührung wird die Methode *dispatchTouchEvent()* aufgerufen, der die Berührung als ein *MotionEvent* übergeben wird. Die Methode *dispatchTouchEvent* liefert einen boolschen Wert zurück ob das Event abgehandelt wurde. Entspricht der Wert *false* so wird das *MotionEvent* an das von der *Activity* referenzierte *Window* und somit der *View* Hierarchie übergeben. Dort wird die CallBack Methode des zugehörigen EventListener Interface des berührten *View* aufgerufen. Im Gegensatz zur Darstellung traversiert ein InputEvent die ViewHierarchie vom berührten Kindknoten bis kein Vaterknoten mehr zur Verfügung steht. Wird die *Activity* auf dem Bildschirm dargestellt wird bei der Zeichnung der *View* Objekte bei der Wurzel begonnen danach folgen die *View* Kindknoten.

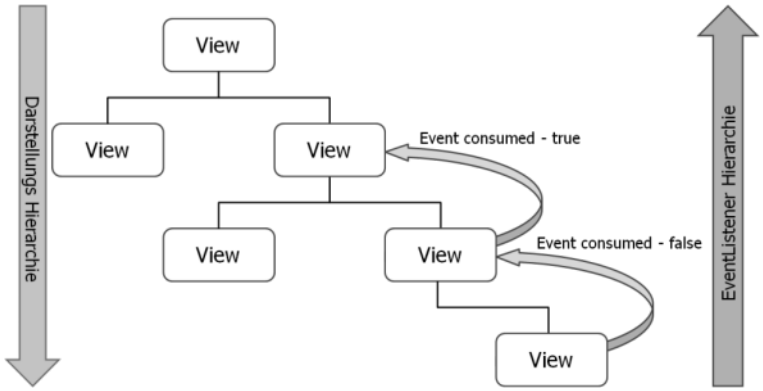

Abbildung 4.9: Skizze der Android Touch Hierarchie - Abhandlung eines MotionEvents

Der OnTouchListener

Handelt es sich bei dem InputEvent um eine Berührung so wird das *OnTouchListener* Interface angesprochen und die Callback Methode *onTouch()* aufgerufen. Die Parameter der Eingabe werden in einem *MotionEvent* zusammengefasst und der Methode übergeben. Wird das *MotionEvent* durch die Methode vollständig abgehandelt liefert die Methode den boolschen Rückgabewert *true* um somit zu verhindern das das Event in der Hierarchie weiter nach oben gereicht wird. Der Hierarchie Baum wird also solange traversiert bis das Event abgehandelt wurde, oder kein weiterer Parent View zur Verfügung steht.

Das MotionEvent

Das *MotionEvent* fasst alle Parameter einer Berührung zusammen. In einem *MotionEvent* können verschiedene Typen von Eingaben erfasst werden, diese werden anhand der *SOURCE_ CLASS* Konstante unterschieden. *MotionEvents* die vom *OnTouchListener* bearbeitet werden sind stets vom Typ *SOURCE_ CLASS_ POINTER*. Dieser umfasst Touch und Mulittouch Geräte. Neben dem auf das Display ausgeübten Druck, der Zeit die seit Beginn der Geste verstrichen ist, der relativen Position zum berührten *View* und der absoluten Position zum Touchscreen auf einer X und Y Achse wird die Art der Berührung erfasst und anhand einer Integer Id, der sogennanten Action, identifiziert. Eine Berührung beginnt stets mit einem *ACTION_ DOWN* Wert und endet entweder mit *ACTION_ UP* oder *ACTION_ CANCEL*. Bei einem Multitouch Display wird für jede zusätzliche Berührung ein *ACTION_ POINER_ DOWN Wert* übermittelt. Jede Berührung verfügt über eine Pointer Id anhand derer Sie identifiziert werden kann, so können

die unterschiedlichen Berührungen in einem *MotionEvent* gespeichert werden. Da alle Eingabewerte einer Berührung in nur einem *MotionEvent* erfasst werden, verfügt dieses über eine Historie, die wie ein LIFO Stack aufgebaut ist, um auf vorangegange Werte zuzugreifen. Dies ist notwendig um die einzelnen Positionen der Berührung abzurufen und so eine Geste zu erfassen.

Die onInterceptTouchEvent Methode

Ein Spezialfall tritt ein wenn es sich bei dem berührten *View* um einen *ViewGroup*, also um einen *View* der mehr als einen Kindknoten haben kann z.B. ein *ListView* oder *GridView*, handelt. Die Klasse *ViewGroup* besitzt zusätzlich die Methode *onInterceptTouchEvent(MotionEvent)*, diese Methode erlaubt das abfangen aller auftretenden Touch InputEvents dadurch ist es möglich *MotionEvents* erst am Vaterknoten zu verarbeiten bevor sie an den Kindknoten weitergereicht werden. Tritt am Gerät eine Eingabe auf, so wird das dazugehörige *MotionEvent* zuerst an die Methode *onInterceptTouchEvent* geliefert. Liefert diese Methode den Wert *true* zurück so wird das *MotionEvent* an den *OnTouchListener* der *ViewGroup* gereicht und nicht an die Kindknoten. Am berührten KindKnoten wird dadurch lediglich ein *MotionEvent* mit der Action *ACTION_CANCEL* empfangen. Damit das *MotionEvent* nicht unnötigerweise durch die *View* Hierachie traversiert, empfielt Google immer einen *OnTouchListener* zu implementieren, falls mit dieser Methode gearbeitet wird.[2]

Der GestureDetector

Um eine Eingabe von Gesten zu erkennen bringt Android seit Version 1 bereits die *GestureDetector* Klasse mit sich. Einem *GestureDetector* Objekt können *MotionEvents* übergeben werden um sie auf Gesten zu überprüfen. Der *GestureDetector* prüft die Übereinstimmung mit einer Geste, mit Gesten aus einer ihm übergebenen *GestureLibrary*. Dabei errechnet er aus der Übereinstimmung der Punkte und Vektoren einen sogenannten Score der angibt wie sehr sich die beiden Gesten gleichen, und liefert eine Liste der erkannten Gesten aus *GestureLibrary* zurück.

Der *GestureDetector* ist aus mehreren Gründen für ein TouchMenü ungeeignet. Da die Erkennung der Geste erst durchgeführt wird wenn die Eingabe der Geste abgeschlossen wurde lassen sich Auswirkungen auf das Menü erst nach der Erkennung anzeige. Beim Vergleich mit der *GestureLibrary* wird die Geste skaliert, dadurch lassen sich statische Menüobjekte nicht addressieren. Aus diesen Gründen wurde auf den Einsatz des *Gestu-*

reDetector verzichtet.

5 Entwicklung des Menüs

5.1 Analyse

Eine Klassenbuch-App Showcase für Tablets soll erstellt werden, die Lehrkräfte bei der Vor- und Nachbearbeitung, aber auch während des Unterrrichts unterstützt. Um die Lehrkräft während des Unterrichts sinnvoll zu unterstützen, muss die Interaktionszeit mit der App zu diesem Zeitpunkt möglichst gering gehalten werden. Fokussiert sich die Lehrkraft zu lange auf das Tablet, ist Sie unaufmerksam der Klasse gegenüber. Eine App zur Unterstützung von Lehrkräften ist also nur sinnvoll wenn die Aufmerksamkeit die man dem Gerät schenken muss klein genug ist um nicht den Kontakt zur Klasse zu verlieren. Mit Hilfe eines Kontext-Menü Konzepts soll dies ermöglicht werden. Das Kontext-Menü soll mit einer einzigen fliessenden Bewegung bedienbar sein um Unterbrechungen zu vermeiden und um die Bedienung mit nur einer Hand, beispielsweise im Stehen, zu ermöglichen. Um zu belegen ob das Kontext-Menü Konzept bei der Interaktion mit der App herkömmlichen Bedienkonzepten, in Bezug auf die Interaktionzeit, überlegen ist. Wird das Konzept mit zwei weiteren Bedienkonzepten, im Rahmen eines Usability Tests verglichen.

5.1.1 Vision

Um eine schnelle und intuitive Eingabe zu ermöglichen soll ein Menü anhand eines Kontext-Menü Konzepts erstellt werden, das es ermöglicht direkt mit Objekten auf dem Touchscreen zu interagieren. Das resultierende Touchmenü soll jedem darstellbaren Objekt zuweisbar sein. Durch berühren eines solchen Interaktionselements wird ein Popup mit dem Hauptmenü eingeblendet. Jedes Menü wird kreisförmig um die Mitte des Interaktionselements angeordnet. Endet die Berührung so wird das aktive Menü ausgeblendet. Wird ein Menüelement berührt wird dieser aktiv. Besitzt er zusätzlich ein Submenü oder ein Popup, so wird dieses eingeblendet. Wird das Interaktionselement berührt und kein Menüpunkt so wird das Hauptmenü eingeblendet. Dies ermöglicht eine Navigation durch das Kontext-Menü.

Durch die kreisförmige Anordnung der Menüpunkte soll die Interaktionszeit optimiert werden. Sie soll es dem Benutzer ermöglichen nach einer kurzen Lernphase ohne erhöhte

Aufmerksamkeit durch das Menü zu navigieren, da sich der Benutzer an die Position erinnert. Er muss also nur die Position des Interaktionselement erfassen und sich die für den gewünschten Menüpunkt nötige Richtung in Erinnerung rufen.

5.1.2 Konzept-Sketch

Ein erster Sketch wie das Kontext Menü angewendet auf eine Klassenbuch App aussehen könnte. Hierbei wird das Submenü zuätzlich zum aktuellen Menü eingeblendet, dadurch entsteht eine sichtbare Hierarchie. Aus Platzgründen wurde darauf in der finalen Version verzichtet, das Vorgängermenü wird stattdessen verkleinert.

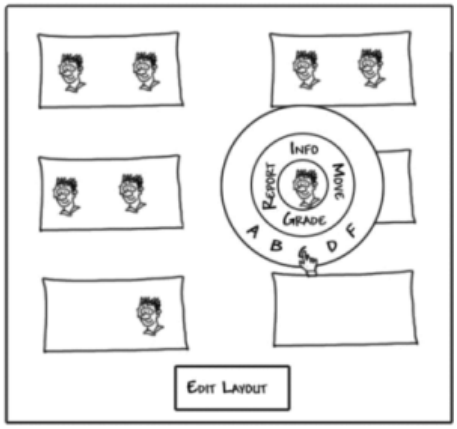

Abbildung 5.1: Konzept-Sketch Kontext Menü

5.1.3 Anforderungsanalyse

An das TouchMenü wurden folgende Anforderungen gestellt:

- Funktionale Anforderungen
    - Primäre Anforderungen - Obligatorisch
        - Das Touchmenü wird als Android Library Objekt angelegt und kann jedem Android Projekt ab Version 3.0 API Level 11 hinzugefügt werden.
        - Das Touchmenü besteht aus einem Menü-Popup welches mindestens ein oder mehr Menüs enthält.

- Jedes Menü kann ein oder mehrere Menüelemente enthalten.
- Jedes Menüelement kann ein Menü referenzieren, welches im Menü-Popup registriert ist. Da jedes Menü wieder Menüelemente enthalten kann wird die Hierarchie-Tiefe dynamisch bestimmt.
- Die Menü-Objekte werden zur Laufzeit generiert und können ähnlich den Swing GUI Elemente verschachtelt werden.
- Objekten vom Typ *View* werden mit dem Menü-Popup als Interaktions-Elemente verknüpft werden.
- Die Anzeige des Menü-Popups beginnt mit der Berührung eines Interaktions-Elements und endet mit dem Aufheben der Berührung.
- Die Anzeige erfolgt unabhängig von der *View*-Hierarchie und das Menü-Popup wird zentriert über dem Interaktions-Element eingeblendet
- Wird das Interaktions-Element berührt das auf das aktive Menü-Popup referenziert wird falls vorhanden das vorhergehende Menü angezeigt, so lässt sich die Menü-Hierarchie auch invers traversieren.
- Ein berührtes Menüelement wird aktiv gesetzt und mittels Farbänderung Feedback an den Benutzer geliefert.
- Jedes MenüElement kann ein *OnItemTouchedListener*-Objekt referenzieren.
- Wird ein Menüelement berührt wird die Methode *onItemTouched()* des *OnItemTouchedListener*-Objekts aufgerufen
- Ist ein MenüElement aktiv wenn eine Berührung endet so wird die Methode *onItemPressed()* des *OnItemTouchedListener*-Objekts aufgerufen.

II Sekundäre Anforderungen - Obligatorisch

- Die Größe des Menü-Popups wird dessen Konstruktor übergeben, alle Menüs und Menü-Elemente werden prozentual zur Größe des Menü-Popups skaliert.
- Wurde ein Interaktions-Element berührt und das Menü-Popup eingeblendet kann das Menü-Popup nur durch das Beenden der Berührung geschlossen werden. Berührungen von weiteren Interaktions-Elementen werden ignoriert.
- Das Interaktions-Element bleibt während der Anzeige des Menü-Popups sichtbar.

- Menüelemente werden kreisförmig um die Mitte des Menü-Popup angeordnet.
- Menüelemente können ein Icon Bitmap-Objekt im Format PNG oder JPG aufnehmen. Das übergebene Bitmap-Objekt wird auf die Größe des Menüelements skaliert.
- Das Standard Menü nimmt mindestens 7 Menü Elemente auf.
- Jedes Menüelement kann ein *View* Objekt aufnehmen um ein zusätzliches Informations-Popup zum Menüelement anzuzeigen.
- Implementierung eines Menüs zur Auswahl eines numerischen Werts [1] aus einem übergebenen Wertebereich.
- Implementierung eines Menüs das eine dynamische Anzahl an Menü-Elemente aufnehmen kann.
- Behandlung von Single- und Multitouch-Eingaben.

III Tertiäre Anforderungen - Optional

- Um eine Bedienung im Stehen zu ermöglichen muss das Touchmenü mit nur einer Hand bedienbar sein.
- Das Menü-Popup wird mittels einer Animation eingeblendet die das Menü-Popup zentriert von einem Wert $< 100\%$ auf 100% skaliert.
- Farb- und Shaderauswahl von Menü sind über Methoden modifzierbar.
- Farb- und Shaderauswahl von aktive und inaktive Menüelementen sind über Methoden modifzierbar.
- Menü-Elemente können neben einem Icon auch einen Kurztext von bis zu 5 Zeichen aufnehmen der angezeigt wird falls kein Icon vorhanden ist.

- Nichtfunktionale Anforderungen

- Um eine intuitive Bedienung zu ermöglichen muss das Touchmenü auf Änderungen der Berührung zügig reagieren, größere Latenzzeiten müssen vermieden werden um den Fluß der Bewegung aufrecht zu erhalten. Da andernfalls der Eindruck einer vollständigen, durchgängigen Geste nicht aufrecht erhalten werden kann.
- Flacker-Effekte beim Wechsel zwischen Menüs durch das Hinzufügen und Entfernen von *View*-Objekten sind zu vermeiden.

[1] Für Ganzzahlige- und Fließkommawerte

5.2 Entwurf

5.2.1 Klassendiagramm

Eine Klassendiagramm Übersicht über die TouchMenu Library.

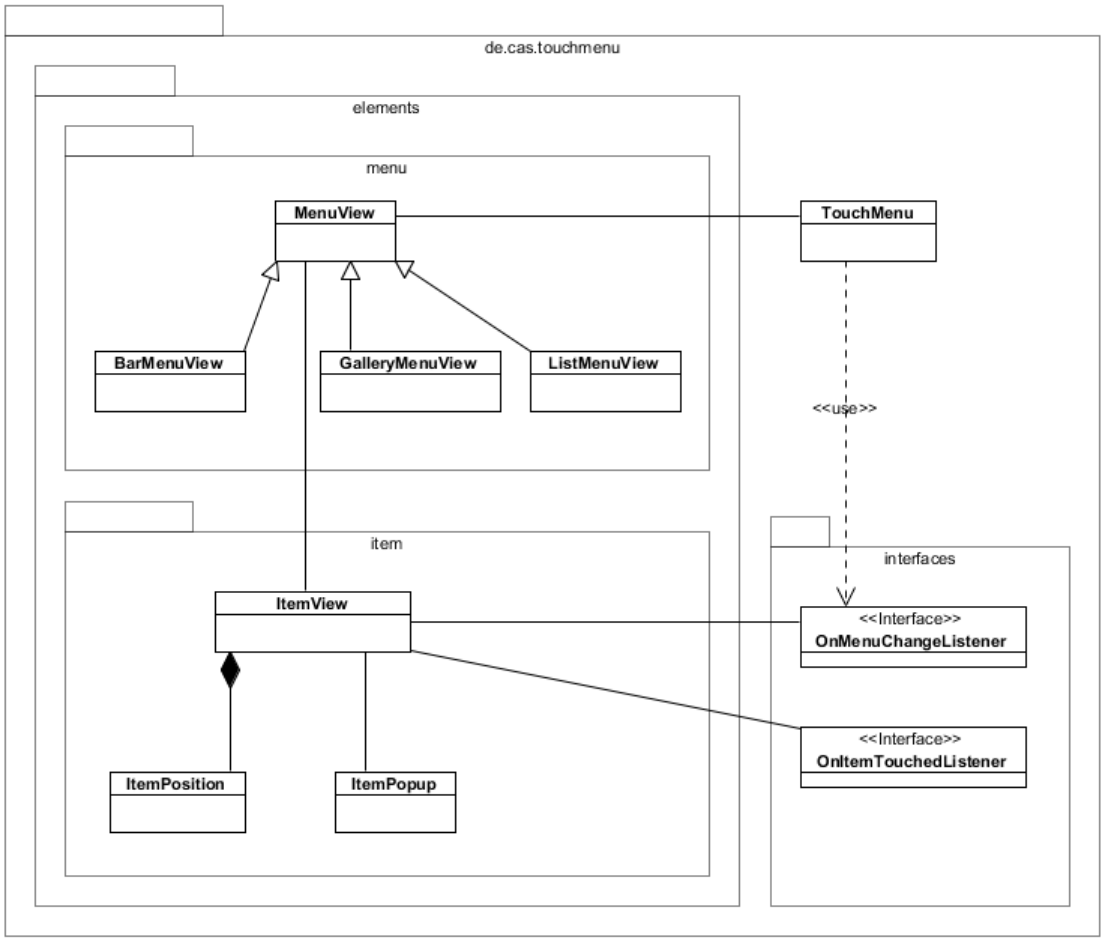

Abbildung 5.2: Klassendiagramm TouchMenu Library

5.2.2 Menü Lifecycle

Der Lifecycle des Menüs wird durch die Berührung bestimmt. Für jede erkannte Berührung wird ein *MotionEvent* angelegt das von TouchListenern verarbeitet werden kann. Die Verarbeitung der MotionEvents wird im nachfolgenden Aktivitätendiagramm dargestellt.

5.2.3 Anwendungsfälle und Sequenzdiagramme

5.3 Implementierung der Touchmenü Library

5.3.1 Android Popup

Das TouchMenü wurde als Subklasse zu *PopupWindow* angelegt. Dadurch wird ist es nicht an die Darstellung durch die Android *View* Hierarchie gebunden. Und kann innerhalb der Grenzen der aufrufenden *Activity* angezeigt werden. Das TouchMenu implementiert zusätzlich den *onTouchListener* der die Position eines *MotionEvents* auf das *PopupWindow* translatiert. Um das Menü aufzurufen und die *MotionEvents* die am Interaktions-Element empfangen werden an das TouchMenü zu übergeben muss das TouchMenü als OnTouchListener beim Interaktions-Element gesetzt werden. Dieses muss daher vom Typ *View* oder einer Subklasse sein.

5.3.2 Menüs und Icons

FlowerMenu

FlowerBarMenu

FlowerListMenu

FlowerItem

Im Konzept Sketch wurde eine das Menü als Kreis um ein Interaktionsobjekt dargestellt. Da es schwierig ist die Position eines Punktes innerhalb eines zweidimensionalen Körpers zu bestimmen wurde stattdessen runde Menüpunkte kreisförmig um das Interaktionselement angeordnet. Dies hat den Vorteil durch Translation der Berührung und Betrag des Vektors der Geste zu errechnen ob ein Menüpunkt berührt wurde.

5.3.3 Implementierug des TouchListeners

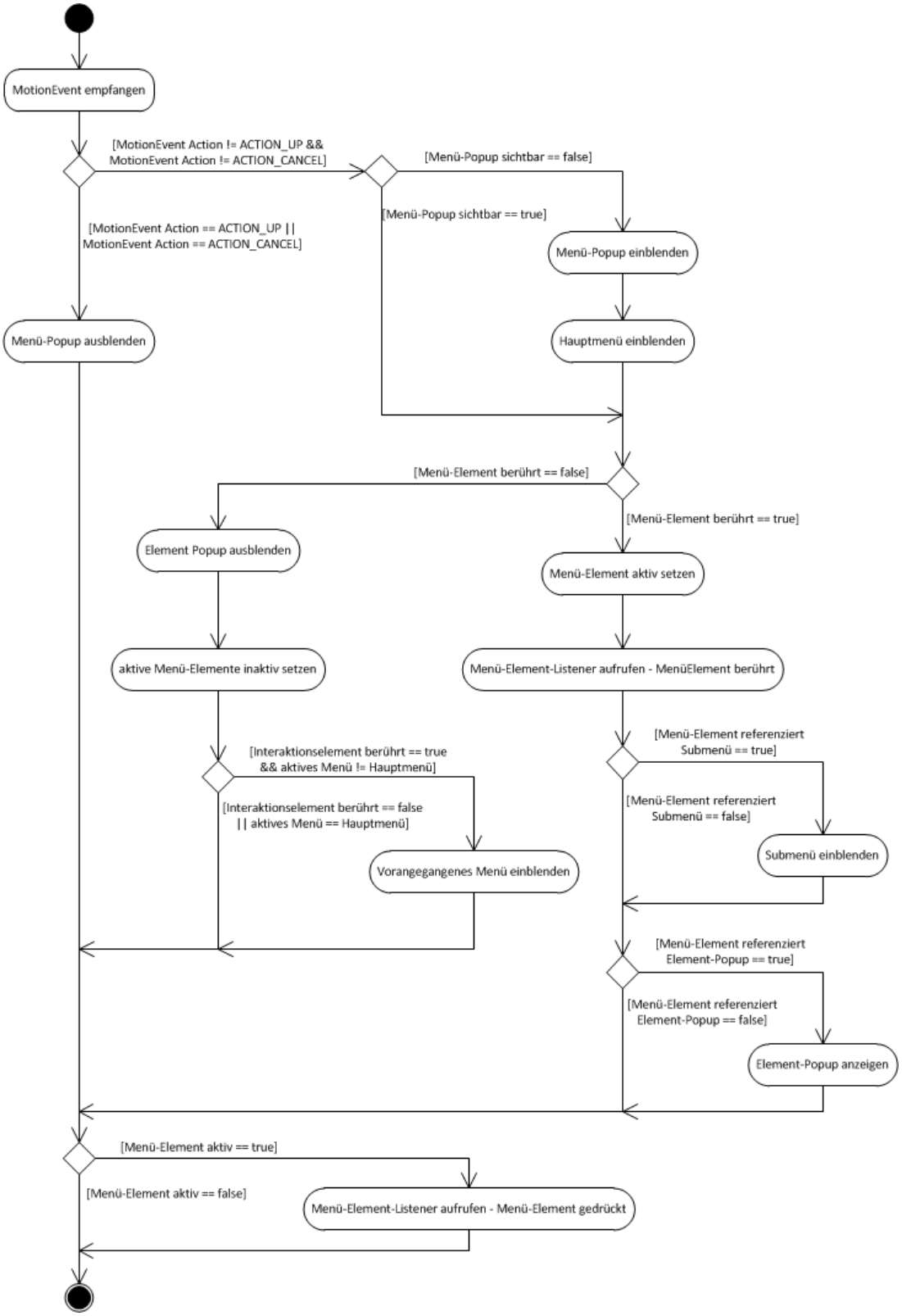

Abbildung 5.3: Aktivitätsdiagramm Abhandlung eines MotionEvents

6 Usabilitytest der Bedienkonzepte

Die Bedienkonzepte werde im Rahmen eines Usability Tests auf folgende Ziele Überprüft: Effektivität des Konzepts Effizienz des Konzepts Sicherer Umgang mit dem Konzept Nützlichkeit des Konzepts Einfache Erlernbarkeit des Konzepts Erinnerung des Konzepts [6]

6.1 Festlegen der Prüfparameter

6.2 Beschreibung der zu prüfenden Bedienkonzepte

6.2.1 Das konservative Konzept

6.2.2 Das App Konzept

6.2.3 Das TouchMenü Konzept

6.3 Ergebnissauswertung

7 Fazit

Literaturverzeichnis

[1] Gesture research – data analysis@ONLINE, Apr 2010.

[2] Google. Android developers reference@ONLINE.

[3] Hayes. Definition gestures - encyclopedia americana.

[4] Christian Kahle. Tablets: Marktanteile von android-systemen sinken@ONLINE, Sep 2011.

[5] Axel Mulder. Hand gestures for hci @ONLINE, Feb 1996.

[6] Jenny Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley Sons, New York, 2007.

[7] Andy Rubin. Where's my gphone @ONLINE, May 2007.

[8] Carol Sinead. Tablets using microsoft software top rim's touted playbook@ONLINE, Jul 2011.

[9] Luke Wroblewski. Touch gesture reference guide@ONLINE, Apr 2010.