TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Forst-, Geo- und Hydrowissenschaften Institut für Kartographie

Diplomarbeit

zum Thema

„Analyse, Konzeption und Entwicklung einer mobilen Kartenanwendung auf Basis des Wanderkalenders der Sächsischen Zeitung“

Betreuer:
Prof. Dr.-Ing. Dirk Burghardt
Dipl.-Ing. Stefan Hahmann

Ausgehändigt am:
15. November 2010

Eingereicht am:
28. April 2011

Eingereicht von:
Eva Hauthal
Matrikel-Nr.: 3203526
Jahrgang 2005

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Forst-, Geo- und Hydrowissenschaften Institut für Kartographie

Aufgabenstellung für die Diplomarbeit

Studiengang: Kartographie

Name der Diplomandin: Eva Hauthal

Thema: Analyse, Konzeption und Entwicklung einer mobilen Kartenanwendung auf Basis des Wanderkalenders der Sächsischen Zeitung

Zielsetzung:

Kartographische Anwendungen auf mobilen Endgeräten besitzen großes Forschungs- und Entwicklungspotential, da sie in Erweiterung zu Papierkarten die Verknüpfung von Interaktivität und Mobilität zulassen. Ziel der Diplomarbeit ist die Konzeption und prototypenhafte Umsetzung einer interaktiven, mobilen Kartenanwendung für touristische Aktivitäten in der Pilotregion Sächsische Schweiz.
Im theoretischen Teil der Arbeit erfolgt ein Literaturstudium zur Forschung im Kontext mobiler Kartographie. Schwerpunkt liegt auf der angepassten, personalisierten Informationsvermittlung, sowie der Einbeziehung nutzergenerierter Inhalte in dynamisch generierte Kartenanwendungen. Es zu analysieren, inwieweit eine Lücke zwischen kartographischer Theorie und technologischer Entwicklung besteht. Im Ergebnis wird ein kartographisches Kommunikationsmodell für mobile, interaktive Karten abgeleitet.
Der praktische Teil der Arbeit basiert auf der Analyse existierender touristischer Applikationen z.B. Lonley-Planet, Handytour, Camineo; welche für die Nutzung auf mobile Endgeräten wie iPhone oder Android-Handys entwickelt wurden. In Anlehnung an den Wanderkalender der Sächsischen Zeitung wird eine eigene mobile Kartenapplikation konzipiert. Anschließend erfolgt die prototypenhafte Umsetzung für das Andorid-Handy G2 Touch. Es ist zu untersuchen, inwieweit existierende Softwarekomponenten genutzt werden können. Als Entwicklungsumgebung ist vorzugsweise das Android-SDK für Eclipse zu verwenden.
Einzureichen sind zwei gedruckte Textfassungen und die digitale Fassung der Arbeit. Die digitale Fassung der Arbeit soll den Textteil und sämtliche für den Prototyp benötigte Daten und Software beinhalten. In einem Übergabegespräch soll die Anwendung dem Betreuer der Hochschule erläutert werden. Die Publikation des Textteils der Arbeit auf dem Publikationsserver Qucosa der SLUB wird angestrebt. Außerdem ist ein Farbposter anzufertigen, auf dem die wichtigsten Ergebnisse der Arbeit zusammengefasst werden.

Betreuer: Prof. Dr.-Ing. Dirk Burghardt
Dipl.-Ing. Stefan Hahmann

Ausgehändigt am: 15.11.2010
Einzureichen am: 15.05.2011

Prof. Dr. Manfred F. Buchroithner
Prüfungsausschuss

Prof. Dr.-Ing. Dirk Burghardt
Betreuender Hochschullehrer

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADT	Android Development Tools
API	Application Programming Interface
APK	Android Package
AVD	Android Virtual Device
BLOB	Binary Large Object
CC/PP	Composite Capability/Preference Profiles
GIS	Geoinformationssystem
GPS	Global Positioning System
GPX	GPS Exchange Format
GUI	Graphical User Interface
HCI	Human Computer Interaction
ID	Identification
IDE	Integrated Development Environment
IS	Infomationssystem
IT	Information Technology
Jar	Java Archive
KIS	Kartographisches Informationssystem
KML	Keyhole Markup Language
KMZ	KML-Datei im Format ZIP
LBS	Location Based Services
LoD	Level of Detail
OGC	Open Geospatial Consortium
PC	Personal Computer
PDA	Personal Digital Assistant
PNG	Portable Network Graphics
POI	Point of Interest
RDF	Resource Description Framework

SDK	Software Development Kit
SMS	Short Message Service
SQL	Structured Query Language
UGC	User Generated Content
VGI	Volunteered Geographic Information
XML	Extensible Markup Language

1. Motivation

Der ununterbrochen wachsende Markt an Mobiltelefonen, Smartphones, Laptops, Notebooks, Navigations- und weiteren mobilen Geräten zeigt, dass die Gegenwart zu Recht als Zeitalter der Kommunikation bezeichnet wird. Einen bedeutenden Beitrag dazu leistete das Aufkommen des mobilen Internets am Ende des vergangenen Jahrhunderts. Von da an waren Informationen und Kommunikationsmöglichkeiten nicht nur am ‚heimischen Rechner' verfügbar, sondern überall und jederzeit. Diese ständige Verfügbarkeit ist für viele Menschen zu einem wichtigen Bestandteil des täglichen Lebens geworden (REICHENBACHER, 2003) und führt dazu, dass verstärkt Karten als Schnittstelle zu mächtigen Informationssystemen sowie als Präsentationsform raumbezogener Informationen angeboten und auch genutzt werden. Obwohl der Aufruf und die Anzahl von Karten im Internet steigen, entstehen diese größtenteils ohne Beachtung der traditionellen Kartographie und deren Methodenlehre und Theorie, d.h. es ist eine Zunahme von ‚*quick and dirty*' erstellten kartographischen Ausdrucksformen zu verzeichnen, die fundamentale kartographische Grundprinzipien missachten. Dieser Qualitätsverlust liegt darin begründet, dass durch einfache und intuitive Benutzeroberflächen, öffentliche APIs und *Mashups* die Kartenerstellung auch für Laien möglich geworden ist. Es wird deutlich, dass gegenwärtig eine Ambivalenz zwischen traditioneller Kartographie und der Popularität von neuen technologischen Anwendungen besteht (GARTNER/SCHMIDT, 2010).

Kurze Zeit nach dem Aufkommen des mobilen Internets wurde das Positionierungsverfahren GPS für die zivile Nutzung freigegeben; dies eröffnete neue Möglichkeiten und ebnete den Weg für die Ära der mobilen Kartographie. Es war nun möglich geworden, den aktuellen Standort des Benutzers in die Kartendarstellung einfließen zu lassen. Gleichzeitig sah sich die Kartographie neuen Herausforderungen gegenüber, da es bei Geräten mit einem sehr kleinen Bildschirm, wie Smartphones oder Navigationsgeräten, inadäquat ist, die gleichen Karten wie für Geräte mit großen Displays zu verwenden. Den technischen Schranken, die durch die Eigenschaften dieser mobilen Endgeräte wie die eben erwähnten kleinen Bildschirmgrößen, geringe Prozessorleistungen oder begrenzte Batterielaufzeiten bestimmt werden, aber auch dem durch die Mobilität der Benutzer hervorgerufenen veränderten Nutzungskontext muss in Form einer Adaption der Karten Rechnung getragen werden durch Anpassung der dargestellten Informationen an den augenblicklichen Bedarf des Benutzers. Andere Schlagworte für diese zentrale Eigenschaft mobiler Karten sind Individualisierung, Personalisierung, Benutzermodellierung, egozentrische Karten oder benutzerzentriertes Design (ZIPF, 2003). Dabei einfließende kontextuelle Parameter werden einerseits durch das mobile Endgerät definiert, andererseits durch den Benutzer selbst. Idealerweise sollten bei letzterem die räumlich-zeitliche Situation, Interessen, Aufgabenkontext, aktuelle Umstände, Ziele, Bedürfnisse etc. Berücksichtigung finden. Damit soll eine Informationsüberflutung des Benutzers vermieden und nur tatsächlich benötigte Informationen in der mobilen Karte visualisiert werden, um so den begrenzten Platz auf den kleinen Displays optimal zu nutzen. Die Vision der mobilen Kartographie ist es, dem Benutzer die räumlich relevanten Informationen zu präsentieren, die er in dieser Situation benötigt. Des Weiteren muss der Grad der Interaktionsmöglichkeiten sowie die Art der Interaktion den mobilen Endgeräten und der mobilen Nutzung angepasst werden. Zwar stellen die möglichen Interaktionen in jedem Fall eine Erweiterung zur statischen Papierkarte dar, im Vergleich zu interaktiven Internetkarten eines Desktop-PCs ist jedoch eine Interaktionsabnahme deutlich erkennbar.

Eine weitere Veränderung, die mit der Mobilität des Kartennutzers einhergeht, ist die sich wandelnde Rolle des Kartographen. Einerseits mobile Endgeräte, die die Nutzung von GPS möglich machen, andererseits die Allgegenwärtigkeit des Internets lassen den Benutzer problemlos eigenständig räumliche Daten erheben und veröffentlichen. Diese Art von Daten werden als nutzergenerierte Inhalte oder *User Generated Content* bezeichnet. Durch diese Bereitschaft vieler Benutzer, freiwillig Informationen zur Verfügung zu stellen, hat sich der Begriff *wikification* etabliert. Dadurch bedürfen traditionelle kartographische Kommunikationsmodelle des Fachbereiches der theoretischen Kartographie, die auf dem Sender-Kanal-Empfänger-Modell basieren, hinsichtlich der mobilen Kartographie einer grundsätzlichen Weiterentwicklung, da eine strikte Trennung in Kartenhersteller und Kartennutzer nicht mehr vorgenommen werden kann. Eine ebensolche Weiterentwicklung ist für kartographische Gestaltungsrichtlinien, die der oben beschriebenen Situation und den gegenwärtigen technologischen Möglichkeiten Rechnung tragen, unumgänglich.

Was KRAAK und ORMELING (2010) in der Geoinformatik als eine Lücke zwischen technologischen Möglichkeiten und kartographischer Theorie (siehe Abbildung 1.1) bezeichnen, die es zu schließen gilt, kann ebenso in der modernen Kartographie wahrgenommen werden.

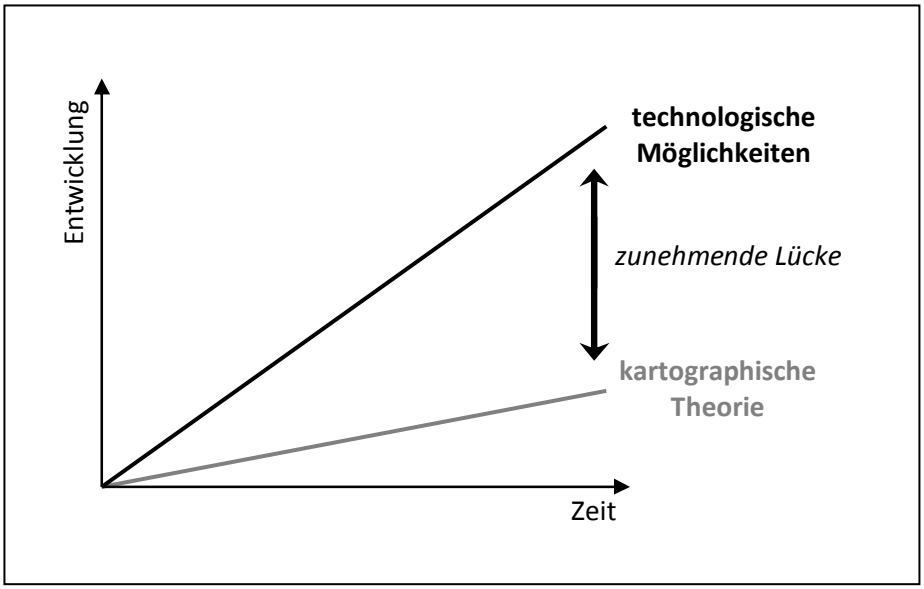


Abb. 1.1: Zunehmende Lücke zwischen technologischen Möglichkeiten und kartographischer Theorie (nach KRAAK/ORMELING, 2010)

2. Definitionen und Grundlagen

2.1 Mobile Kartographie

Der Terminus ‚mobile Kartographie' bedeutet nicht, dass Karten mobil, im Sinne von transportabel sind; das waren sie von Anbeginn ihrer Zeit. Vielmehr ist gemeint, dass die Karten mobiler Endgeräte auf die Mobilität des Benutzers reagieren und sich eben dieser mobilen Nutzung anpassen können. Dies geschieht durch die Bestimmung des aktuellen Standortes des Benutzers, der sich durch Positionierungsverfahren wie GPS ermitteln lässt. Dabei wird gleichzeitig der Zeitpunkt der Informationsnutzung definiert, welcher zusammen mit dem Standort den räumlich-zeitlichen Kontext der Nutzung und des Benutzers bildet. Anhand dieses Kontexts wird die Geoinformation auf diese konkrete Nutzungssituation adaptiert durch Auswahl, Aufbereitung und Präsentation (REICHENBACHER/ANGSÜSSER/MENG, 2002).

Ein erster Definitionsversuch dieses relativ neuen Forschungsfeldes wurde von REICHENBACHER (2001) unternommen:

„Mobile Kartographie beschäftigt sich mit Theorien und Technologien der dynamischen kartographischen Visualisierung von räumlichen Daten und ihrer interaktiven Nutzung auf mobilen Endgeräten überall und zu jeder Zeit unter spezieller Berücksichtigung des tatsächlichen Kontexts und der Eigenschaften des Benutzers."

Somit charakterisiert der Begriff ‚mobil' hier drei wesentliche Komponenten (REICHENBACHER/ ANGSÜSSER/MENG, 2002):

- das mobile Medium
- die mobile Nutzung
- der mobile Inhalt, der sich dynamisch an die mobile Nutzungssituation anpasst

2.2 Adaption

Adaption ist die aus dem Lateinischen stammende Bezeichnung für den Begriff ‚Anpassung' und wird in Wissenschaft und Technologie mannigfach verwendet. Im Falle eines Computersystems bedeutet Adaption die Anpassung von Parametern an den Benutzer. Dabei kann zwischen Adaptierbarkeit und Adaptivität unterschieden werden (REICHENBACHER, 2003).

Die Adaption von Services und Inhalten umfasst hauptsächlich ein Filtern von verfügbaren Informationen. Dieser Filterungsprozess basiert auf einem Benutzerprofil oder -modell, das die Interessen, Fähigkeiten und Charakteristiken des Benutzers beschreibt (POSLAD ET AL, 2001).

2.2.1 Adaptierbarkeit

Ein System besitzt die Eigenschaft der Adaptierbarkeit, wenn es auf der Grundlage einer extern vorgenommenen Diagnose durch extern vorgenommene Eingriffe so eingestellt werden kann, dass es den Aufgabenstellungen und dem Kenntnisstand des Benutzers möglichst gut entspricht. Somit ist Adaptierbarkeit eine Form der Individualisierung von Benutzerschnittstellen und eine Vorbereitung der Systemnutzung, die vom System nicht permanent angeboten wird. Im Gegensatz zu adaptiven Systemen liegt die Initiative und Ausführung der Systemveränderung auf eine interaktive Art und Weise in der Hand des Benutzers (BOLLMANN/KOCH, 2001; REICHENBACHER, 2003).

2.2.2 Adaptivität

Auch Adaptivität ist eine Form der Individualisierung von Benutzerschnittstellen. Hier erfolgt die Initiative und Ausführung einer Systemanpassung hingegen durch das System selbst. Somit gilt ein System als adaptiv, wenn es in der Lage ist, weitestgehend automatisiert und ohne umfangreiche Eingriffe des Benutzers, dessen Unterstützungsbedarf zu diagnostizieren und das Ergebnis dieser Diagnose in geeignete Aktionen umzusetzen (BOLLMANN/KOCH, 2001; MEYER, 2010).

2.3 Benutzermodellierung

Hierbei handelt es sich um einen Modellierungsprozess, der das Ziel hat, ein System an einen spezifischen Benutzer oder eine Benutzergruppe zu adaptieren. Grundlage für die Modellierung sind Merkmale des Benutzers wie Erfahrungen mit dem System, Nutzungshäufigkeit, Wissen in der Problemdomäne, Präferenzen der Interaktionsformen sowie Aufgabenstellung und -kontext (BOLLMANN/KOCH, 2001).

2.4 Kontext

‚Kontext' ist ein Wort, das viele verschiedene Bedeutungen haben kann - abhängig vom Kontext.

Eine der am meisten aufgegriffenen Definitionen stammt von DEY und ABOWD (1999). Ihr zufolge ist jede Information, die genutzt werden kann, um die Situation einer Entität zu beschreiben, ein Kontext. Eine Entität kann hierbei eine Person, ein Ort oder ein Objekt sein, das als relevant erachtet wird für die Interaktion zwischen dem Benutzer und einer Anwendung, einschließlich dem Benutzer und der Anwendung selbst.

Weiterhin unterscheiden die Autoren zwischen primärem und sekundärem Kontext. Primärer Kontext umfasst die Elemente Standort, Zeit, Identität und Aktivität. Diese können als Indizes der Elemente des sekundären Kontexts fungieren, wie zum Beispiel Verabredung, Wetter, Bedingungen.

In einem ähnlichen Ansatz von VAN WELIE und DE RIDDER (2001) stimmen drei Elemente mit denen von DEY und ABOWD (1999) überein: Identität, Standort und Zeit. Das vierte Element ist das Gerät. Es

beschreibt die technischen Eigenschaften des mobilen Endgerätes, welche für die Informationsübermittlung und -darstellung von großer Bedeutung sind.

2.5 Interaktion

Der Begriff ‚Interaktion' stammt ursprünglich aus den Sozialwissenschaften und meint das ‚miteinander-in-Verbindung-Treten' mehrerer Elemente (BOLLMANN/KOCH, 2001) oder auch ‚wechselseitig aufeinander bezogene menschliche Handlungen' (QUIRING/SCHWEIGER, 2006). Im Bereich der Mensch-Computer-Interaktion sind dies Aktionen, die vom Benutzer am Computer ausgeführt werden können. Tastatur, Maus oder Touchscreen stellen verschiedene Interaktionstechniken dar (BOLLMANN/KOCH, 2001). Aus Software-technischer Sicht sind Interaktionen mit dem Computer eine Form der Informationssuche oder Dateneingabe (SCHULMEISTER, 2002).

Interaktivität ist demzufolge eine Eigenschaft von Computersystemen, dem Benutzer verschiedene Eingriffs- und Steuermöglichkeiten im direkten Dialog mit dem Computer zur Verfügung zu stellen. Hierbei bestimmt der Grad der möglichen Steuerung das Maß der Interaktivität des Computersystems (BOLLMANN/KOCH, 2001).

2.6 Mensch-Computer-Interaktion

Die Mensch-Computer-Interaktion (engl. *Human-Computer-Interaction* (HCI)) stellt die Gesamtheit derjenigen Wechselwirkungen zwischen dem Menschen und Datenverarbeitungssystemen dar, die erforderlich sind für ein bestimmungsgemäßes und fehlerfreies Funktionieren von Computersystemen (BOLLMANN/KOCH, 2001). Das Forschungsgebiet der Mensch-Computer-Interaktion hat sich zur Aufgabe gemacht, die Nutzerschnittstellen, die die Interaktion zwischen Mensch und Maschine verbessern bzw. erleichtern sollen, zu gestalten sowie sich die Nutzerfreundlichkeit bzw. *Usability* von Hard- und Software zum Ziel gesetzt (QUIRING/SCHWEIGER, 2006).

2.7 Kartographische Kommunikation

Unter kartographischer Kommunikation werden ein- oder mehrseitige Übertragungsprozesse bei der Aufnahme, der Verarbeitung und dem Austausch von raumbezogenen Informationen mittels Karten oder anderen kartographischen Medien auf der Grundlage eines gemeinsamen Zeichenvorrates verstanden. Die kartographische Kommunikation verfolgt als Ziele die georäumliche Erkenntnisgewinnung bzw. -erweiterung, eine raum- und umweltbezogene Bewusstseinsbildung und die Steuerung von Verhalten und Handeln im Raum. Übermittelt werden georäumliche Informationen, kommunikationsrelevante Merkmale sowie Bedingungen der Aufnahme, Verarbeitung oder des Austauschs von Informationen (BOLLMANN/KOCH, 2001).

2.8 User Generated Content

Der Begriff *User Generated Content* (UGC) wird im Deutschen als ‚nutzergenerierte Inhalte' oder ‚nutzergeschaffene Inhalte' bezeichnet. Weniger häufig verwendete englischsprachige Synonyme sind *User Driven Content* oder *Consumer Created Content*.

Es gibt verschiedene Ansätze für die Definition dieses Begriffes, jedoch keine allgemein gültige. Eine Definition hinsichtlich des Web 2.0 beschreibt *User Generated Content* anhand von drei Merkmalen (STÖCKL/GRAU/HESS, 2006):

- Der Benutzer nimmt eine Doppelfunktion ein. Er übernimmt sowohl die Rolle des passiven Inhaltskonsumenten als auch die des aktiven Inhaltsproduzenten. Die durch ihn erstellten Inhalte werden in den meisten Fällen nicht redaktionell aufbereitet.
- Der Produzent von *User Generated Content* beabsichtigt primär keine monetäre Vergütung, eine spätere finanzielle Nutzung der Inhalte ist jedoch nicht ausgeschlossen.
- Die Adressierung von *User Generated Content* erfolgt an eine große Anzahl von Rezipienten. Im Gegensatz zu anderen Massenmedien wird die Mehrzahl der Beiträge aber nur vereinzelt wahrgenommen.

User Generated Content ist kein in sich geschlossener Begriff, sondern vereint eine Anzahl unterschiedlicher Plattformen, Dienste und Darstellungsformen und stellt eine Sammlung verschiedener Medien dar. Diese können, wie in Tabelle 2.1 dargestellt, in unterschiedliche Ausprägungen und Funktionalitäten unterteilt werden (KOLBITSCH/MAURER, 2006).

Darstellung	**Art der Inhalte**	**Beispiele**
Private Weblogs	Text, Fotos	*Bildblog*, *Dresden-Blog*
Wikis	Text	*Wikipedia*, *Wikinews*
Podcasts	Audio, Video	*Chaosradio*, *Elektrischer Reporter*
Soziale Netzwerke	Text, Video, Foto	*Facebook*, *YouTube*, *Flickr*, *Twitter*
Social Bookmarks / *Folksonomies*	Text, Hyperlinks	*Delicious*, *Mister Wong*

Tab. 2.1: Unterteilung von User Generated Content nach Darstellung und Art der Inhalte (nach KOLBITSCH/MAURER, 2006)

Dieses Phänomen sozialen Verhaltens, dass Tausende Menschen bereit sind, viel Zeit zu investieren, raumbezogene Inhalte im Internet zu teilen, ohne Aussicht auf finanzielle Belohnung und oft ohne

jegliche Gewissheit, dass diese Inhalte tatsächlich benötigt werden, wird von GOODCHILD (2007b) als *Volunteered Geographic Information* (VGI) bzw. *Volunteered Geography* bezeichnet.

2.9 Location-based Services

Location-based Services (LBS) sind IT-Services an drahtlosen, mobilen digitalen Kommunikations- und Informationsendgeräten, die Informationen zur Verfügung stellen, welche unter Berücksichtigung des aktuellen Standortes des Benutzers oder des mobilen Objektes erstellt, kompiliert, ausgewählt oder gefiltert wurden. Damit zählt LBS zu den sogenannten *context-aware Services* (KÜPPER, 2005).

Aus technischer Sicht kann bei der Informationsübermittlung zwischen *Push*- und *Pull-Services* unterschieden werden. Ein *Push-Service* liefert dem Benutzer ohne Rückfrage Informationen. Bei einem *Pull-Service* hingegen bekommt der Benutzer nur Informationen übermittelt, wenn er diese anfordert (SAYDA, 2006).

2.10 Smartphone

Smartphones sind Mobiltelefone mit einem erweitertem Funktionsumfang. Sie sind daher nicht nur für Telefonie und *Short Message Servic*e (SMS) optimiert, sondern bieten eine wesentlich größere Bandbreite an Anwendungsmöglichkeiten, wie Navigation, Funktionalitäten zur Aufnahme und Wiedergabe audiovisueller Inhalte sowie zur Navigation oder Zusatzdienste wie Email. Des Weiteren sind Smartphones oft mit Sensoren ausgestattet, die für herkömmliche Mobiltelefone untypisch sind. Solche Sensoren können GPS-Empfänger, Magnetfeld-, Lage-, Bewegungs- oder Lichtsensoren sein. Ein Smartphone kann als PDA, also als ein kleiner transportabler Computer, mit Mobiltelefonfunktionalität angesehen werden.

Im Gegensatz zu klassischen Mobiltelefonen laufen auf Smartphones komplexe Betriebssysteme, die in der Mehrzahl über eine offene API verfügen. Dadurch kann der Benutzer auch Programme von Drittanwendern installieren. Auf dem Markt haben sich verschiedene Betriebssysteme für Smartphones etabliert (SJURTS, 2011):

- *Symbian*
- *BlackBerry von RIM*
- *Apple iOS*
- *Linux* (in verschiedenen Ausgestaltungen: *Android*, *webOS*, *MeeGo*, *Mobilinux* und *Openmoko*)
- *Windows Phone* von Microsoft bzw. die Vorgängerversion *Windows Mobile*
- *bada*
- *Brew*

Diese Betriebssysteme ermöglichen es, weitere Applikationen zu installieren und somit das Smartphone individuell vom Benutzer mit neuen Funktionen auszustatten.

Abbildung 2.1 zeigt die Verteilung der weltweiten Smartphone-Marktanteile des Jahres 2010 unterteilt nach Betriebssystem.

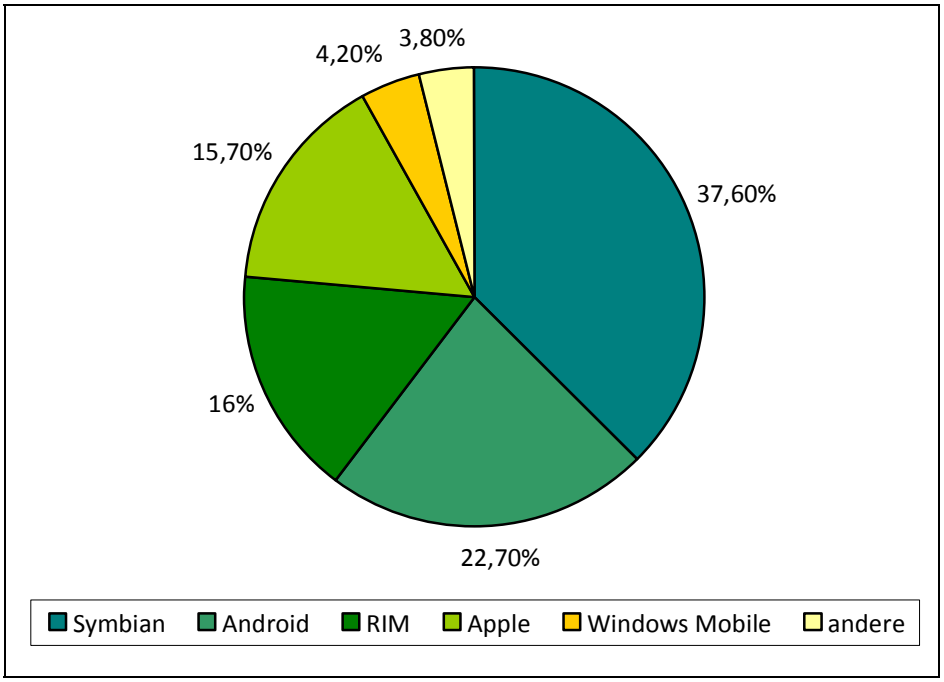


Abb. 2.1: Verteilung der weltweiten Smartphone-Marktanteile für 2010 nach Betriebssystem (GARTNER NEWSROOM, 2011)

2.11 Applikation

Auch App (kurz für *application*, engl.). Im Allgemeinen sind Applikationen auf Informationssystemen ausgeführte Anwendungsprogramme oder Dienste jeglicher Art.

Mobile Applikationen sind Anwendungen, die auf mobilen Endgeräten über drahtlose Kommunikationsnetzwerke, in diesem Fall speziell über das Mobilfunknetz, abgewickelt werden. Mittlerweile sind damit im Sprachgebrauch meist Anwendungen für moderne Smartphones gemeint, die in der Regel über einen *Onlineshop*, der in das Betriebsystem des Gerätes integriert ist, bezogen und als Zusatzsoftware installiert werden können, aber auch Anwendungen, die sich fest installiert und nicht löschbar auf dem Gerät befinden, zählen dazu (SJURTS, 2011).

3. Angepasste, personalisierte Informationsübermittlung in der mobilen Kartographie

Die Realisierung des drahtlosen Internetzugangs hat Web-Karten wieder in die mobile Umgebung zurückgebracht; also dorthin, wo sie am meisten benötigt werden (MENG/REICHENBACHER, 2005). Mit dieser ‚Vereinigung' der realen und virtuellen Welt gehen zahlreiche Herausforderungen für den Kartographen einher. Das minimalistische Display der Geräte erfordert eine Adaption der Inhalte und der Darstellung an die augenblicklichen Anforderungen und kognitiven Fähigkeiten des Benutzers. Nicht nur technische Faktoren wie die begrenzte Displaygröße oder Batterielaufzeiten, sondern auch nichttechnische Faktoren wie zeitabhängige Benutzeraufgaben oder die sich ständig verändernde Umgebung des Benutzers zwingen den Kartographen, nur wirklich benötigte Informationen und Interaktionen in mobile Karten einzubetten.

Da eine mobile Karte bereits bekannte Eigenschaften neu kombiniert, kommt ihr ein bemerkenswertes Potential zu. Dies ist in Tabelle 3.1 erkennbar, wobei davon ausgegangen wird, dass die digitalen Karten der letzten beiden Spalten immobil an einem herkömmlichen Desktop-PC genutzt werden (REICHENBACHER/ANGSÜSSER/MENG, 2002).

	Digitale mobile Karten	**Analog oder digital erstellte gedruckte Karte**	**Digitale Karten – offline (CD-ROM)**	**Digitale Karten – online (Web)**
Medium				
mobil	✓	✓	-	-
positionierbar	✓	-	-	(✓)
chronometrisierbar	✓	-	-	✓
Inhalt				
dynamisch	✓	-	✓	✓
multimedial	✓	-	✓	✓
adaptierbar	✓	-	✓	✓
adaptiv	✓	-	✓	✓
interaktiv	✓	-	✓	✓
ortsabhängig gestaltbar	✓	-	(✓)	(✓)
zeitabhängig gestaltbar	✓	-	(✓)	✓
Nutzung				
mobil	✓	✓	-	-
synchron	✓	-	-	✓
ortsunabhängig	✓	✓	-	-
zeitunabhängig	✓	✓	✓	(✓)
Nutzgs-Sit. rasch wechselnd	✓	✓	-	-
ressourcenbeschränkt	✓	-	-	-

Tab. 3.1: Vergleich der Eigenschaften mobiler und anderer Karten (nach REICHENBACHER/ANGSÜSSER/MENG, 2002)

Die Entwicklung einer mobilen Karte kann als benutzerzentriertes Design realisiert werden. Laut ISO 13407 ist benutzerzentriertes Design ein iterativer Prozess, der aus mehreren Phasen besteht (MENG/REICHENBACHER, 2005):

1) Identifizieren des Bedarfs nach einem benutzerzentrierten Design
2) Spezifizierung des Benutzers und seiner organisatorischen Anforderungen
3) Erstellen von Designlösungen
4) Evaluierung des Designs gegenüber den Anforderungen

In der mobilen Kartographie entspricht die erste Phase der generellen und oben genannten Erkenntnis, dass nur eine Auswahl an Informationen dem Benutzer präsentiert werden kann. In der zweiten Phase findet eine Erkennung und Modellierung des Benutzers und seines Kontexts statt. Die Adaption als Folge dieser Vorgänge stellt die dritte Phase dar. Eine Evaluierung findet während der praktischen Anwendung in diesem Sinne nicht statt, wobei eine Anpassung einer bereits adaptierten Karte durch den Benutzer, indem er die Eigenschaft der Adaptierbarkeit nutzt, im Moment der Verwendung als Evaluation gegenüber seinen Anforderungen angesehen werden kann. Im Folgenden soll speziell auf die zweite und dritte Phase eingegangen werden.

3.1 Kontext

Um eine Anwendung an verschiedene Bedürfnisse anzupassen, die in mobilen Nutzungssituationen entstehen, muss der Kontext des mobilen Benutzers verstanden werden.

Wie in der Definition in Kapitel 2.4 bereits deutlich wurde, umfasst dieser Terminus ein sehr weites Feld. Neben den schon erwähnten Facetten des Kontexts gibt es des Weiteren physische Dimensionen, Systemdimensionen, semantische und soziale Aspekte. Im Bereich der mobilen Kartographie stellt der Kontext vielmehr ein generelles Konzept dar, das eher spezifische Dimensionen wie Situation, Benutzer, Aktivitäten, Information und System enthält. Diese Dimensionen sind keineswegs unabhängig voneinander, sondern durch teilweise sehr komplexe Beziehungen miteinander verbunden, welche in Abbildung 3.1 aufgezeigt werden. Die größte Schwierigkeit besteht darin, aus dieser enorm umfangreichen Kontextmenge die Parameter herauszufiltern, die für die mobile Kartographie relevant sind und einen signifikanten Einfluss auf die mobile georäumliche Informationsverwendung haben (REICHENBACHER, 2003).

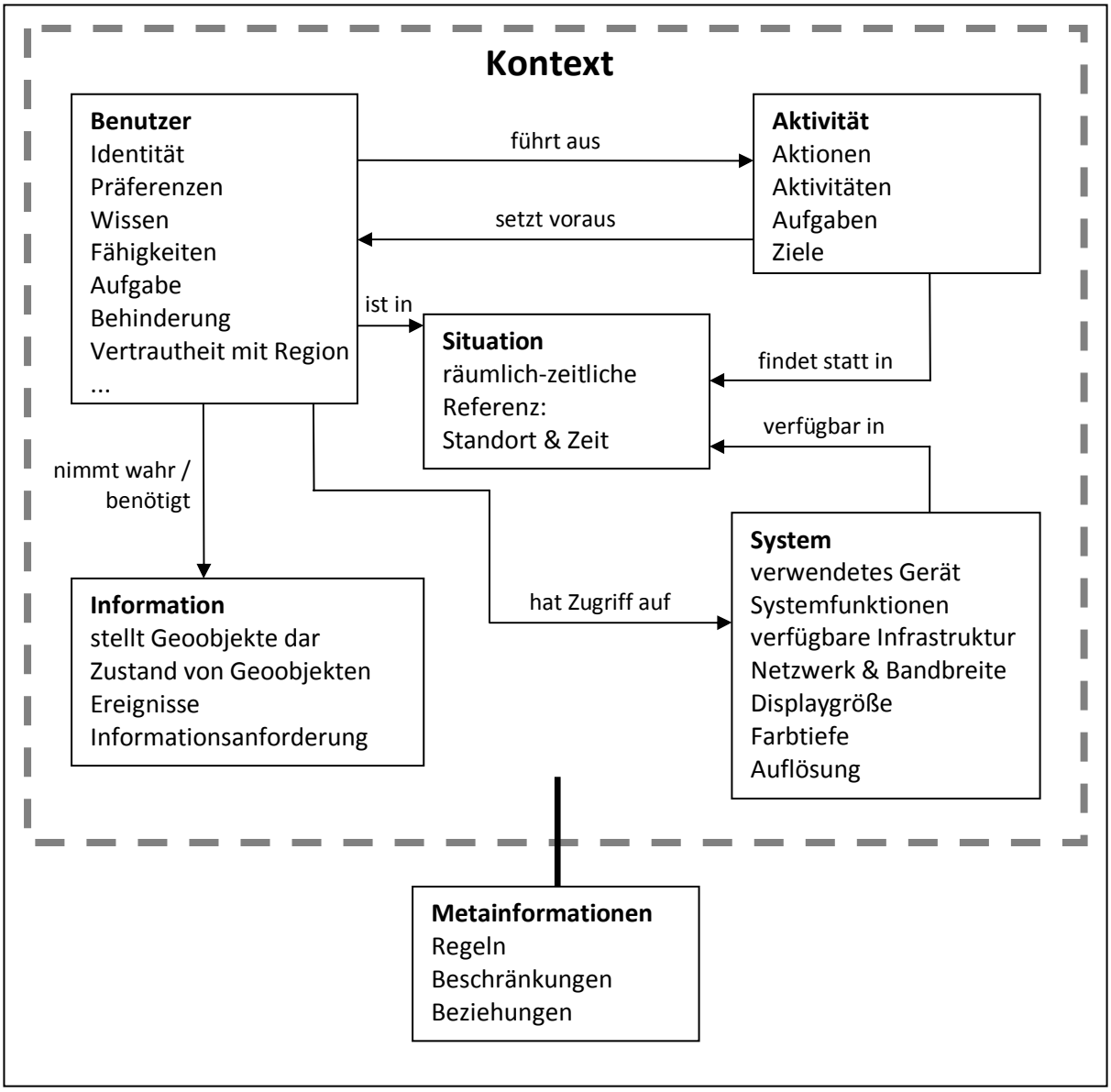


Abb. 3.1: Kontext in der mobilen Kartographie (nach REICHENBACHER, 2003)

3.1.1 Kontextdimensionen

In Abbildung 3.1 ist erkennbar, dass die Situation des Benutzers innerhalb eines Kontexts den Kern dieses Gesamtgebildes darstellt und eine Funktion von Standort und Zeit ist. Situation wird hier in ihrem ursprünglichen Sinne verstanden (lat. *situs*) als situiert, d.h. platziert sein in einem räumlich-zeitlichen Referenzsystem, das auf den restlichen Kontext ausgerichtet ist. Gemeint ist demnach nicht die Situation als Umstand oder Umwelt. Die Situiertheit des Benutzers hält den Kontext bereit. Die Ursache dafür, hier zwischen Situation und Kontext zu unterscheiden, liegt darin, dass (im Sinne von LBS) Adaption direkt an der Situation, also an Standort und Zeit, ausgerichtet werden kann oder in einer umfassenderen Manier an einem Kontext.

Die Information des Standortes ist an die Position des Benutzers gebunden und unterscheidet verschiedene Stufen der Granularität (z. B. Punktkoordinate, Adresse, Ortsname, Region) mit

unterschiedlichen Wertebereichen. Die für spezifische Aktivitäten oder Informationsbedürfnisse benötigte Granularität variiert stark. Natürlich bedarf die Mehrzahl von Diensten keiner Genauigkeit im Millimeter- oder Zentimeterbereich, sondern von einigen Metern. Neben dem Standort ist auch die Orientierung des Benutzers ein wichtiger Kontextfaktor. Die Orientierung des Gerätes kann mittels elektronischem Kompass oder kinematischem GPS bestimmt werden.

Für die Zeitinformationen existieren die gleichen Granularitätsunterschiede wie für den Standort. Zeit kann beispielsweise eine exakte Systemzeit repräsentieren, aber auch eine Tages- oder Jahreszeit. So sehen Orientierungspunkte wie Gebäude auf Grund von Beleuchtungsunterschieden bei Nacht anders aus als am Tag. Diese zeitliche Granularität beeinflusst die Reaktion des Systems oder des Dienstes auf Benutzeranfragen. Natürlich muss ebenso die geographische Information als Pendant zeitlich modelliert werden.

Der Benutzer und auch die Aktivitäten, die er ausführt, sind situiert, daher die verschiedenen Dimensionen des Kontexts. An dieser Stelle stößt man unvermeidlich auf den Begriff der Benutzermodellierung, welche hauptsächlich im Internet für die Adaption von Inhalten Anwendung findet, hier jedoch Teil des Kontextmodells ist. Genauso wie bei anderen Dimensionen stellt sich die Frage, welche Charakteristika des Benutzers modelliert werden müssen bzw. welche Eigenschaft wichtig und relevant für die mobile geographische Informationsverwendung und ihre Adaption sind. Darauf wird näher in Kapitel 3.1.3 eingegangen.

Laut Definition sind Aktivitäten in einen Kontext eingebettet und sind somit Teil dieses Kontexts. In einem anfänglichen Stadium ist es angemessen, eine gradlinige mobile Aktivitätstaxonomie aufzustellen. Es ist also wichtig, für einen gegebenen Kontext zu wissen, welche Aktivitäten möglich, erlaubt, geeignet etc. sind. Dies gehört zu der komplizierten Aufgabe, Regeln und Vorgaben für einen Kontext zu modellieren. Ein anderer Aspekt bezüglich der Aktivitäten ist die soziale Dimension, beispielsweise die Tatsache, ob eine Aktivität alleine oder in einer Gruppe ausgeführt wird, ob andere Personen involviert oder betroffen sind. Diese Informationen können aber auch in der Benutzerdimension modelliert werden.

Auch die Information stellt in der mobilen Kartographie eine Kontextdimension dar, die den Kontext der Objekte innerhalb des Kontextbereiches beschreibt, wie zum Beispiel Gebäudetypen. Diese Art an kontextuellen Informationen werden als kollokierte georäumliche Objekte bezeichnet. Ein gutes Beispiel für Kollokation ist die Gegenwart einer Kirche, die die Gegenwart eines Friedhofes indiziert. Diskrete georäumliche Objekte können mit diesem Ansatz der Kollokation modelliert werden, d.h. die Präsenz eines georäumlichen Objektes kann mit einer gewissen Wahrscheinlichkeit oder Mutmaßlichkeit durch die Untersuchung eines anderen Objektes zurückgestellt werden. Da die Eigenschaften und Zustände dieser kollokierten georäumlichen Objekte den Kontext beträchtlich charakterisieren, sind sie von höchstem Interesse. Diese Informationen überschneiden sich aber teilweise mit einem anderen Typ der kontextuellen Information, den physischen Parametern (z.B. Temperatur, Feuchtigkeit, Lärmpegel etc.).

Eine weitere wichtige Dimension des Kontexts, die keinesfalls unterbewertet werden darf, sind die Charakteristika des benutzten Systems. Geräte- und Netzwerkeigenschaften haben einen wesentlichen Einfluss darauf, wie Informationen am besten übertragen und visualisiert werden

können. Ein Standard zur Beschreibung von Gerätekapazitäten ist *Composite Capabilities/Preference Profile* (CC/PP). CC/PP ist ein Protokoll, dass vom World Wide Web Consortium definiert wurde und den *Resource Description Framework* (RDF) beschreibt.

Letztendlich ist es von großer Wichtigkeit, die relevanten Kontextdimensionen zu identifizieren und die Beziehungen zwischen ihnen zu modellieren sowie Regeln und Vorgaben, die für einen Kontext gelten, zu definieren. Beispielsweise hat der physische Kontext (wie das Wetter) Einfluss auf mögliche Aktivitäten (z.B. ist Schlittschuhlaufen auf einem See im Sommer nicht möglich). Andererseits bilden auch Aktivitäten einen Kontext, wie Bus fahren oder Shopping. Auf diese Art kann eine Ausschmückung der Aktivitätentypologie helfen, eine entsprechende Kontexttypologie zu definieren (REICHENBACHER, 2003).

3.1.2 Kontextmodellierung

Die oben erläuterten Kontextinformationen können teilweise aus verschiedenen Sensorinformationen abgeleitet, teilweise können sie aus anderen Quellen bezogen und müssen nicht *in situ* gemessen werden. Sowohl das Internet als auch Webservices, die ein *Distributed Computing Environment* darstellen, erlauben das Extrahieren von Kontextinformationen durch Georeferenzierungsmethoden. Ein Beispiel hierfür ist das Abrufen von Witterungsbedingungen für einen gegebenen Standort von einem meteorologischen Webservice.

Für jegliche Art der Adaption ist die Formalisierung des Kontexts essentiell. Dabei werden Situation und Kontext als eine Funktion mehrerer Parameter beschrieben. Diese Parameter sind verschiedener Art: einige Parameter können numerisch, andere symbolisch sein, weitere sind atomisch und wieder andere stellen eine Zusammensetzung aus Subparametern dar.

Die Situations- (*S*) und Kontextfunktion (*C*) kann wie folgt beschrieben werden:

$S = f_1 (L, t)$ und $C = f_2 (S, U, A, I, T) | R$

wobei *L* = Standort (Position als Koordinatenpaar, Ortsname, Adresse)
t = Zeit (exakter Zeitpunkt, Zeitintervall, Tageszeit)
U = Benutzer (Identität: uid oder Benutzergruppe: gid)
A = Aktivitäten {lokalisieren, navigieren, suchen, identifizieren, überprüfen}
I = Information
T = System (Gerätekapazitäten, Netwerktyp, Bandbreite)
R = Vorgaben und Regeln (gültig für den Kontext)

Das dabei gebildete Modell kann allmählich vom Benutzer über seine Aktivitäten und seine Interaktionen mit dem mobilen System lernen. Dieser Formalismus des Kontexts kann genutzt werden, um Kontextgebrauch oder Gebrauchsmuster in der mobilen Kartographie zu erfassen (REICHENBACHER, 2003).

Stark verbunden mit der Kontextmodellierung ist der Ansatz von SCHMIDT und GELLERSEN (2001) mit dem die Validität des Kontexts bestimmt wird. Sie argumentieren, dass diese von Raum und Zeit ab-

hängig ist. Mit zunehmender räumlicher oder zeitlicher Distanz von der Existenz eines Kontexts nimmt die Validität ab. In diesem Ansatz werden drei Prinzipien unterschieden:

- **Prinzip der Lokalität:** Die Relevanz eines Kontext ist maximal im Ursprung des Standortes und nimmt mit zunehmender Distanz zum Ursprung ab. Ab einer bestimmten Distanz zum Ursprung hat der Kontext keine Relevanz mehr.
- **Prinzip der Temporarität:** Die Relevanz eines Kontext ist maximal für den Zeitpunkt seines Ursprungs und nimmt mit zunehmender zeitlicher Distanz zum Ursprung ab. Ab einer bestimmten Distanz zum Zeitpunkt des Ursprungs hat der Kontext keine Relevanz mehr.
- **Prinzip der Unabhängigkeit:** Der Benutzer und der Erzeuger des Kontexts sind unabhängig voneinander. Mehrere Benutzer und Erzeuger eines Kontexts können gleichzeitig unabhängig existieren.
- **Prinzip der Verteilung und Skalierbarkeit:** Die Verteilung von Informationen ist räumlich gebunden (örtlich skalierbar). Die Existenz von Informationen ist zeitlich gebunden (zeitlich skalierbar).

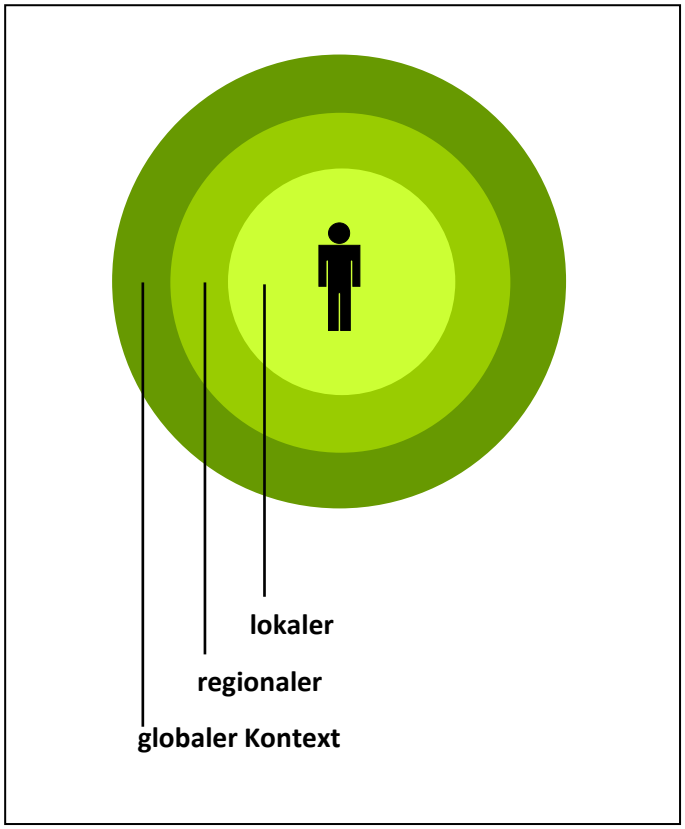


Abb. 3.2: Räumlicher Geltungsbereich des Kontexts
(nach REICHENBACHER, 2003)

SCHMIDT und GELLERSEN (2001) schlagen ein konkretes räumlich-zeitliches Kontext-Relevanz-Modell vor, das sich der *Fuzzy-Set-Theorie*, also der unscharfen Mengenlehre, bedient. Die Grundidee der *Fuzzy-Set-Theorie* ist, dass die Zugehörigkeit von Mengen nicht binär ist, aber dass mit einer bestimmten Relevanz ein Element zu einer Menge gehört. Dies wird durch eine Funktion bestimmt, die Zugehörigkeitsfunktion.

Abgesehen von den einzelnen Kontextdimensionen und ihrer Formalisierung müssen zwei Arten von Kontext zu strukturieren erwähnt werden: der Gültigkeitsbereich des Kontexts und die Niveaus des Kontexts. Ähnlich wie bei mobilen Aktivitäten ist der Kontext, in dem diese Aktivitäten stattfinden, abhängig vom Maßstab. Kontext kann auf verschiedene Maßstäbe zurückgeführt und als lokal, regional oder global angesehen werden (siehe Abbildung 3.2). In diesem Sinne ist der Maßstab notwendig, um Kontextmuster zu erkennen. In der mobilen Kartographie kann Kontext als eine Hierarchie angesehen werden, die aus verschiedenen Ebenen zusammengesetzt ist. Daraus ergibt sich eine Steigung von allgemeinem zu mehr spezialisiertem Kontext (siehe Abbildung 3.3), die es erlaubt, die Modellierung mit dem allgemeinen Kontext zu beginnen, was diesen Vorgang zunächst vereinfacht (REICHENBACHER, 2003).

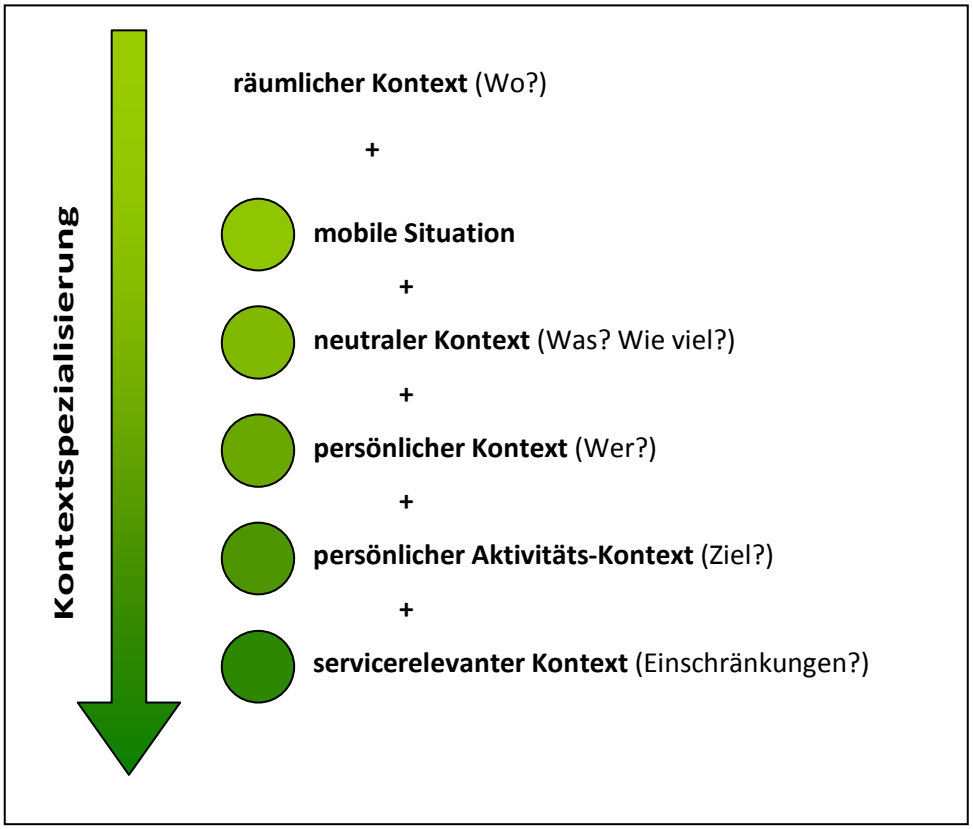


Abb. 3.3: Hierarchische Ebenen des Kontexts (nach REICHENBACHER, 2003)

3.1.3 Benutzermodellierung

Da der Benutzer als eine der Kontextdimensionen einen großen Einfluss auf die Adaption einer mobilen Karte hat, soll an dieser Stelle näher auf seine Modellierung eingegangen werden.

Um Charakteristika eines Benutzers einfließen zu lassen, ist eine Benutzermodellierung erforderlich. Dabei gibt es eine gebräuchliche Vorgehensweise: das System sammelt Daten über den Benutzer, diese werden zu einem Benutzermodell verarbeitet und anschließend für die Adaption genutzt, wie in Abbildung 3.4 dargestellt (HÖLLDOBLER, 2001).

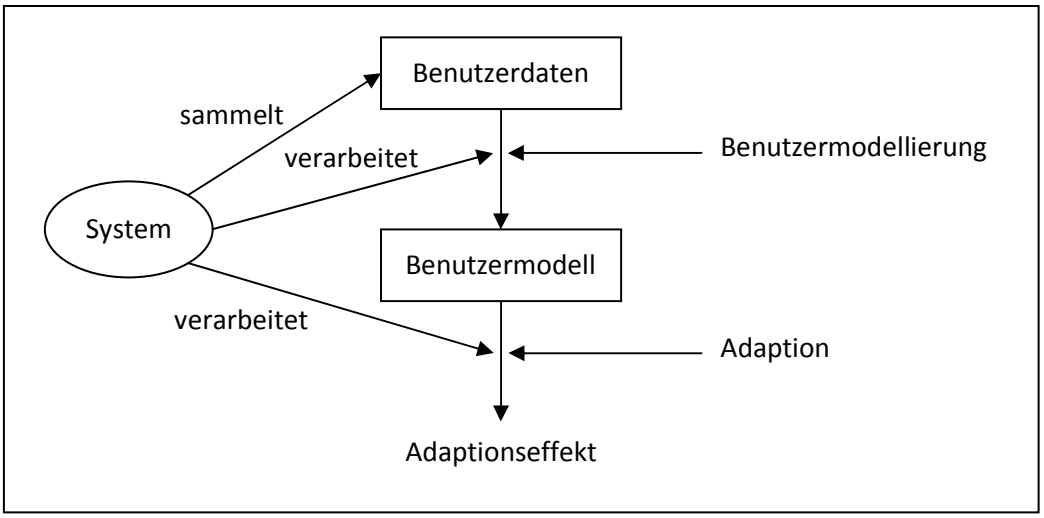


Abb. 3.4: Klassische Schleife ‚Benutzermodellierung - Adaption' in adaptiven Systemen (HÖLLDOBLER, 2001)

Für den ersten Schritt der Datenakquisition stellt sich zunächst die Frage, welche Aspekte des Benutzers für die Adaption relevant sind und in das Benutzermodell einfließen sollen. Zweifelsohne hängt dies von der jeweiligen Anwendung ab, dennoch folgt an dieser Stelle eine Auflistung von Charakteristika, die bei der Geovisualisierung auf mobilen Endgeräten berücksichtigt werden können oder sollten (HÖLLDOBLER, 2001):

- **Ziele/Aufgaben:** Das Ziel hängt meist weniger mit dem individuellen Benutzer als mit dem augenblicklichen Kontext zusammen und kann sich stetig ändern. Oft findet eine Unterscheidung zwischen übergeordneten Zielen und kleineren Unterzielen statt. Die Aufgabe, die mit der Karte gelöst werden soll, spielt demzufolge eine große Rolle. Als typische kartenspezifische Aufgaben bei mobilen Endgeräten können im Wesentlichen Navigation, Orientierung und die Suche bestimmter Lokalitäten genannt werden (ZIPF, 2003).
- **(Vor-) Wissen/Erfahrung:** Hier betrifft dies vor allem die Frage, wie der Benutzer mit der Region, für die er eine mobile Karte benötigt, vertraut ist (*Mental Map*). Diese Eigenschaft kann sich aber ebenso auf bereits gesammelte Erfahrungswerte im Umgang und in der Bedienung des mobilen Gerätes beziehen.

- **Präferenzen:** Diese können entweder über das Verhalten des Benutzers geschlussfolgert oder von ihm erfragt werden. Die Aufgabe des Systems ist es, Vorlieben so weit zu verallgemeinern, dass sie sich auch auf einen anderen Kontext übertragen lassen. Oft werden diese Informationen in numerischer Form repräsentiert, damit Präferenzen verschiedener Benutzer zu Gruppenmodellen zusammengefasst werden können.

Des Weiteren haben die folgenden demographischen, körperlichen und interessenmäßigen Eigenschaften eines Benutzers Einfluss auf Inhalt und Design einer Karte (ZIPF, 2003):

- Alter
- Kulturkreis
- Bildung
- kognitive Fähigkeiten
- sozialer Status
- Geschlecht
- körperliche Einschränkungen
- Interessen
- Gewohnheiten
- Müdigkeit, Fitness
- Hunger, Eile

Als ein Beispiel für die Eigenschaft des Kulturkreises sei zu nennen, dass es in unterschiedlichen Ländern verschiedene Kartenstile und unterschiedliche gebräuchliche Kartensignaturen gibt. Ebenso variiert die Interpretation von Farben deutlich in verschiedenen Kulturen (ZIPF, 2002). Daraus ergibt sich eine unbewusste Erwartungshaltung des Benutzers bezüglich des Aussehens von Karten.

Es ist offensichtlich unmöglich und auch nicht notwendig all die genannten Facetten eines Benutzers in die Darstellung auf einem winzigen Bildschirm einfließen zu lassen. Oftmals ist es ausreichend, den Benutzer anhand weniger Information zu modellieren (MENG, 2005).

Das Hauptproblem bei dem gesamten Prozess der Benutzermodellierung ist die Akquisition der Daten über den Benutzer. Diese können entweder durch das System selbst erfragt (explizite Methode) oder durch eine Beobachtung des Benutzers ermittelt werden (implizite Methode). Letzteres ist zwar weniger exakt, stört oder verärgert den Benutzer jedoch nicht. Eine explizite Befragung des Benutzers ist präziser, doch sie unterbricht seine gegenwärtige Tätigkeit mit dem System, ist zeitraubend und kann Unmut hervorrufen, was die implizite Methode für die Benutzerdatenakquisition befürwortet (POSLAD ET AL, 2001).

Wie bereits erwähnt, besteht die zuletzt genannte Methode darin, die Daten für ein Benutzermodell selbstständig durch Beobachtung zu ermitteln. Diese Methode ist allerdings nur begrenzt einsetzbar,

da das System nur durch die Verfolgung des Benutzerpfades innerhalb der jeweilige Anwendung Informationen gewinnen kann. Daraus ergeben sich Informationen, wie häufig der Benutzer einen Knotenpunkt besucht hat, jedoch ist nicht erkennbar, ob dieser Knotenpunkt tatsächlich benötigt wurde bzw. sich der Benutzer mit den Inhalten beschäftigt hat. Das macht diese Vorgehensweise nicht sehr zuverlässig (HÖLLDOBLER, 2001).

Die zweite Möglichkeit ist die explizite Methode, welche durch ein Nachfragen des Systems vor oder während der Nutzung oder auch benutzergesteuert durch das Erstellen eines Profils von statten geht (MEYER, 2010). Die explizite Methode wird auch kollaborative Benutzermodellierung genannt. Mit diesem Begriff werden in der Literatur verschiedene Methoden bezeichnet. Die einfachste Vorgehensweise ist das Beantworten von Fragen durch den Benutzer. Problematisch ist, dass er hierfür motiviert sein muss (Aufwand, Datenschutz etc.). Da diese Daten automatisch verarbeitet werden, müssen in den Antwortmöglichkeiten Alternativen zur Auswahl angeboten werden, was die Aussagekraft der Antworten aber einschränkt, da so nur eine begrenzte Menge an Attributen beschrieben werden kann. Eine andere Variante ist, dass der Benutzer dem System zeigt, welche Form der Adaption er bevorzugt. Wichtig ist dabei, dass die möglichen Interaktionen für den Benutzer einfach und durchschaubar sind. Dieser Ansatz wurde teilweise dahingehend erweitert, dass der Benutzer selbstständig das Benutzermodell verändern kann. Dafür muss es ihm aber verständlich gemacht werden (HÖLLDOBLER, 2001).

Je nach Ziel und Umfang der Adaption, nach der Art der Informationen, die gesammelt werden sollen und nach Auskunftsbereitschaft des Benutzers hat sowohl die implizite als auch die explizite Methode ihre Vor- und Nachteile. Aus diesem Grund findet in der Praxis oftmals eine Kombination beider Varianten Anwendung (MEYER, 2010).

SCHWAB und POHL (1999) beschreiben eine Akquisitionsmethode, die auf statistischen Signifikanzanalysen basiert. Wenn ein Benutzer eine Information signifikant öfter erfragt als der Durchschnitt, ist er wahrscheinlich an dieser Information interessiert. Die Stärke dieses Ansatzes ist, dass er mit einer definierten Aussagewahrscheinlichkeit (z.B. 99%) Wissen über den Benutzer bereitstellt; eine Schwäche besteht hingegen darin, dass mehrere Interaktionen, mit denen der Benutzer sein Interesse an einer Information bekundet, nötig sind, bevor das System dieses signifikante Interesse voraussetzen kann. Dies kann kompensiert werden durch ein initiales Benutzerprofil in Form von Stereotypen, indem der Benutzer nur wenige Fragen zu Demographie und anderen Indikatoren beantwortet, die es erlauben, ihn zu klassifizieren (POSLAD ET AL, 2001).

Beim anschließenden Aufbau des Benutzermodells gibt es ebenso mehrere Vorgehensweisen. Die traditionelle ist die Verwendung von Stereotypen. Dabei werden Benutzer zu typischen Benutzergruppen zusammengefasst, die über bestimmte Eigenschaften definiert sind. Die Zuordnung eines Benutzers zu einer Gruppe erfolgt anhand von wenigen Informationen zu einen oder mehreren Stereotypen. In der Schlussfolgerung erhält der jeweilige Benutzer die darin definierten Eigenschaften. Die Zuordnung geschieht meist über binäre Werte oder über Wahrscheinlichkeiten. Dieser stereotypische Ansatz ist einfach zu erstellen und zu verwalten, oftmals jedoch nicht detailliert und aussagekräftig genug. Daher wird meist so vorgegangen, dass die Stereotypen zur Initialisierung des

Benutzermodells verwendet werden, die Modellierung des Benutzerverhaltens jedoch mit anderen Techniken vorgenommen wird.

Eine weitere Vorgehensweise ist der Aufbau von Benutzermodellen mithilfe maschineller Lernverfahren. Hierbei werden Beobachtungen über den Benutzer als Trainingsdaten zur Eingabe verwendet, wobei die Historie seiner Interaktionen berücksichtigt wird. Das erlaubt eine flexible Anpassung des Systems an sich verändernde Ansprüche. Die Ausgabe ist eine Unterstützung bei einer Entscheidung, welche meistens eine Klassifikationsmethode darstellt. Diese Vorgehensweise liefert vielversprechende Ergebnisse, aber es gibt zwei Schwachstellen: erstens benötigen die meisten Verfahren Eingabedaten in Form von Negativ- und Positivbeispielen, die aus Informationen über den Benutzer generiert werden müssen und zweitens arbeiten diese Verfahren mit einer sehr großen Datenmenge, die im Bereich der Benutzermodellierung jedoch selten zur Verfügung steht. Aus Letzterem ergibt sich die Fragestellung, welche Algorithmen auszuwählen sind, die mit einer geringen Datenmenge aussagekräftige Ergebnisse erreichen können (HÖLLDOBLER, 2001).

Bei den oben vorgestellten Methoden, handelt es sich um eine Benutzermodellierung im Moment der Systemnutzung. Es gibt darüber hinaus noch die Variante, dass der Benutzer bereits *a priori* modelliert wird. Auch hier kommen verschiedene Methoden zum Einsatz.

Beim sensorbasierten Ansatz trägt der Proband ein *Head-Mounted-Tracking-Gerät*, das während der Interaktion mit der Karte sein Blickverhalten oder seine Gehirnwellen registriert. Die Aufmerksamkeitsverteilung der Kartensymbole hängt direkt mit dieser Bewegungsspur zusammen und enthüllt Wahrnehmungsparameter wie Auffälligkeit und Betrachtungsdauer einzelner Symbole oder Fixationsorte, -häufigkeit, -dauer, und -reihenfolge. Der mentale Aufwand des Benutzers wird hingegen in den Gehirnwellen wiedergegeben. Mithilfe des sensorbasierten Ansatzes kann eine Qualitätssicherung garantiert und Diskrepanzen zwischen den Erwartungen des Gestalters und dem Verhalten des Benutzers aufgedeckt werden.

Mit dem szenariobasierten Ansatz werden sich häufig wiederholende Benutzungsfälle modelliert und eine Untermenge aller Visualisierungswerkzeuge und interaktiver Funktionen identifiziert. Durch eine Analyse einer hinreichend großen Stichprobe von Szenarien lassen sich persönliche Verhaltensmuster eines Benutzers bestimmen.

Bei dem Ansatz der *Repertory-Grids* wird der Proband aufgefordert, sich in einen Benutzungskontext zu versetzen und angebotene Gestaltungslösungen auf vergleichende Weise und mit eigenen Worten zu bewerten. Die Bewertungsergebnisse werden pro Gestaltungslösung in einem sogenannten *Repertory-Grid* gespeichert. Daraus lassen sich neue Bewertungsdimensionen und Anforderungen für Karten ableiten.

Wenn dabei die Sichtweise des Benutzers zu sehr von der des Gestalters abweicht, kann auf die sogenannte teilnehmende Gestaltung zurückgegriffen werden. Wie der Name bereits sagt, nimmt der Benutzer an der Gestaltung der Karte teil. Je früher und umfassender dies geschieht, desto eher können Gestaltungsprobleme entdeckt und bewältigt werden und einen umso größeren Einfluss übt der Benutzer auf die Gesamtlösung aus.

Beim aufgabenbasierten Ansatz bekommt der Proband Aufgaben gestellt, die sich in unterschiedliche Granularitäten unterteilen lassen. Aus der Betrachtung des individuellen aufgabengetriebenen Benutzerverhaltens können interessante Hinweise auf Computerkenntnisse und Gewohnheiten abgeleitet werden (MENG, 2004).

3.2 Adaption

Die Adaption kann als Antwort auf die Veränderung eines Kontexts gesehen werden. Die Einführung des Adaptionsmechanismus in die Kartographie hat das Ziel, dem Benutzer dabei zu helfen, geographische Informationen effizienter zu nutzen, deren Darstellung an seine Bedürfnisse und die begrenzten Ressourcen anzupassen sowie die gesamte Relevanz zu verbessern. Das Ergebnis ist eine adaptive Karte, die hier als kartenähnliche Visualisierung definiert werden kann, welche in einem beträchtlichen Umfang durch den Benutzer oder das System adaptiert wurde. Dabei sind einige grundsätzliche Prinzipien zu beachten (REICHENBACHER, 2003):

- Die Adaption muss transparent sein, d.h. der Benutzer muss wissen, dass eine Adaption stattfindet. Diese sollte, wann immer möglich, in Form von Vorschlägen geschehen.
- Der Benutzer muss immer die gesamte Kontrolle haben, d.h. der Benutzer kann entscheiden, ob er eine Adaption insgesamt oder nur an bestimmten Punkten wünscht.
- Die Adaption muss adaptierbar sein, d.h. wenn es der Benutzer wünscht, kann er jederzeit selbst die Geovisualisierung ändern und zwar im gleichen Ausmaß wie das adaptive Verhalten des Systems.

Für Adaption in der mobilen Kartographie ergeben sich vier verschiedene Domänen, die von ihr betroffen sind (REICHENBACHER, 2003):

- **Informationsdomäne:** Der Informationsgehalt wird adaptiert an die aktuelle Situation, den Benutzer, seine Aktivitäten und das System.
- **Benutzeroberflächendomäne:** Die Benutzeroberfläche wird adaptiert an die Situation, den Benutzer, seine Aktivitäten, das System, die physischen Bedingungen und die Mobilität.
- **Darstellungsdomäne:** Die Visualisierung der Informationen wird adaptiert an die Situation, den Benutzer, seine Aktivitäten, kollokierte Objekte, das System, die physischen Bedingungen und die Mobilität.
- **Technologiedomäne:** Die Informationskodierung wird adaptiert an die spezifischen Geräte mit unterschiedlichen Eigenschaften (Bildschirmgröße und –auflösung, Speicher, Prozessorleistung etc.) oder an die Übertragungsmedien (z.B. Bandbreite).

Die Kartenadaption sollte bei Bedarf auch die Qualität und Güte dieser Domänen einbeziehen bzw. darstellen (ZIPF, 2003).

3.2.1 Adaptionsobjekte

Theoretisch sind alle Eigenschaften und Funktionen, die für das Design und die Verwendung von Geovisualisierung eine Rolle spielen, potentielle adaptierbare Objekte. Diese adaptierbaren Objekte lassen sich den eben erwähnten Domänen zuordnen. Demnach wird die Benutzeroberfläche meist von dem verwendeten mobilen Endgerät bestimmt. Ein PDA mit berührungsempfindlichem Bildschirm erlaubt beispielsweise andere Interaktionen als ein Smartphone mit Tastaturfeld, somit bedarf auch der Interaktionsstil gegebenenfalls einer Adaption. Ebenso kann die Verfügbarkeit von interaktiven Kartenfunktionen oder deren Granularität adaptiert werden sowie der Interaktionsmodus, die geographische Information in verschiedenster Weise und die Visualisierung. Tabelle 3.2 gibt einen Überblick über alle adaptierbaren Objekte und ihren jeweiligen Wertebereich (REICHENBACHER, 2003).

Adaptionsobjekte	**Attribut / Variable / Parameter**	**Wertebereich**
Kartenbestandteile	Selektion	{Attribut | räumlich | zeitlich} Bedingung
	Gruppierung	Bedingung
	Klassifikation	Bedingung
	Geographisches Gebiet	BoundingBox: xmin, ymin, xmax, ymax
Karteninteraktionen	Modalität	{visuell, akustisch, gemischt}
	Stil	{Point-and-Click, Formen, Menüs, Anfrage, natürliche Sprache}
	Modus	{auswählen, verschieben, zoomen, eingeben}
Kartenfunktionen	verschieben	{verfügbar, verborgen, deaktiviert}
	zoomen	{verfügbar, verborgen, deaktiviert}
	Kartengebiet wählen	{verfügbar, verborgen, deaktiviert}
	Karten-Layer wählen	{verfügbar, verborgen, deaktiviert}
	Kartenobjekt wählen	{verfügbar, verborgen, deaktiviert}
	zeigen	{verfügbar, verborgen, deaktiviert}
	Attribute anzeigen	{verfügbar, verborgen, deaktiviert}
	Distanz berechnen	{verfügbar, verborgen, deaktiviert}
	Umfang berechnen	{verfügbar, verborgen, deaktiviert}
	Fläche berechnen	{verfügbar, verborgen, deaktiviert}
	neu entwerfen	{verfügbar, verborgen, deaktiviert}
Karten-Layout	Titel	{Präsenz, Position, Größe}
	Legende	{Präsenz, Position, Größe}
	Maßstabsleiste	{Präsenz, Position, Größe}

Kartenstil	Methoden- / Form- / Grafikstruktur	{Bild, Perspektive, Karte, Topogramm}
	Kartenausschnitt (räumlicher Fokus)	{BoundingBox | Zentrum (x, y), Radius r | Ortsname | Aktivitätenregion }
	Kartenorientierung	Grad
	Kartenmaßstab	Maßstabsfaktor
	Level of detail	LoD
	Generalisierung	Operator, Algorithmus, Parameter, Reihenfolge
Kartengrafik	Dimension	Punkt, Linie, Fläche
	Position	(x, y)
	Farbe	CMYK, RGB, HSV, HEX
	Wert	
	Größe	{mm, px, %}
	Orientierung	Grad
	Form	{Quadrat, Dreieck, Kreis, Ellipse}
	Muster	{...}
	Deckkraft	{{0, 1} | %}
	Klarheit	
Kartentext	*siehe Kartengrafik* +	
	Schriftart	Schriftart-Name
	Schriftschnitt	{normal, fett, kursiv, fettkursiv}

Tab. 3.2: Adaptierbare Objekte und ihr Wertebereich (nach REICHENBACHER, 2003)

3.2.2 Adaptionsmethoden

In Analogie zu den verschiedenen Domänen der Adaption, die zu Beginn von Kapitel 3.2 beschrieben wurden, können Adaptionsmethoden den Informationsgehalt und die Informationskodierung, die Informationsstruktur, die Benutzeroberfläche oder die Informationsdarstellung betreffen. Inhaltliche Adaptionsmethoden adaptieren Kartenbestandteile, strukturelle Adaptionsmethoden adaptieren ebenso Kartenbestandteile sowie -interaktionen oder -funktionen und morphologische Adaptionsmethoden adaptieren den Kartenstil, Graphiken und Text (vgl. Tabelle 3.2). Die folgende Liste gibt einen Überblick über mögliche Adaptionsmethoden für mobile Karten (REICHENBACHER, 2003):

- Auswahl von Kartenbestandteilen
- Reduzierung des Karteninhaltes
- Reduzierung der Informationsdichte
- Entfernen, Weglassen oder Eliminieren von Kartenobjekten
- Priorisieren von Informationen
- Ersetzen oder Austauschen von äquivalenten Präsentationen
- Wechsel zwischen vordefinierten Gestaltungsalternativen
- Ändern von Präsentationen, (Zurück-) Ändern von Symbolisierungen

- (Re-) Konfigurieren von Kartenkomponenten
- Modifizieren der Benutzeroberfläche
- Ändern der Codierung

3.2.4 Adaptionsprozess

Eine detaillierte algorithmische Sichtweise auf den Prozess der Adaption ist in Abbildung 3.5 dargestellt. Der Prozess besteht aus mehreren Schritten. Der erste Schritt ist das Erkennen des Kontexts und somit des Adaptionsziels. Dieses enthält die Elemente, an welche die Visualisierung der geographischen Informationen adaptiert werden soll. Einige Elemente können vom System gemessen, andere (wie beispielsweise Präferenzen) sollten vom Benutzer vordefiniert und in einem Profil abgespeichert werden. Die Adaption wird bei jeder Änderung oder Differenz von Standort, Benutzer, Aktivität, Informationsbedarf oder System ausgelöst sobald sie einen festgelegten Grenzwert überschreitet oder ein bestimmtes Ereignis eintritt, denn eine solche Änderung lässt eine Lücke zwischen Kartenzweck und -design entstehen. Die Kontextparameter werden in die Entscheidungsmaschine eingegeben, welche anhand des identifizierten Kontexts überprüft, ob eine Adaption notwendig ist und wählt gegebenenfalls eine geeignete Adaptionsstrategie. Wenn eine Adaption ausgelöst wird, wird die Adaptionsmaschine aufgerufen, die die geeigneten Methoden mit den entsprechenden Parametereinstellungen auswählt und Regeln aus dem Adaptionsmodell anwendet. Des Weiteren wählt die Adaptionsmaschine das Objekt, das adaptiert werden soll. Den letzten Schritt stellt die Adaptionsausführung dar, d.h. die Konstruktion des Adapters. Dieser Adapter bewirkt die Änderungen am Adaptionsobjekt indem er die gewählten Methoden, Parameterwerte und Regeln anwendet .

Der Entwurf von adaptiven Systemen oder Diensten erfordert mindestens zwei Modelle (REICHENBACHER, 2003):

- **Aufgabenmodell:** Dieses Modell beschreibt die Aufgaben oder Aktivitäten und die sich ändernden Bedürfnisse des Benutzers.
- **Benutzermodell:** Dieses Modell adressiert Benutzerrollen und die zu einer Benutzergruppe gehörenden unterschiedlichen Bedürfnisse für verschiedene Benutzergruppen oder -rollen.

Weitere wichtige Modelle sind (REICHENBACHER, 2003):

- **Domänenmodell:** In diesem Modell wird Wissen über die reale Domäne erfasst.
- **Systemmodell:** Dieses Modell beinhaltet Wissen über das System, d.h. über das Leistungsspektrum des Gerätes, über verfügbare Funktionen etc.
- **Adaptionsmodell:** Das Adaptionsmodell besteht aus den Adaptionsregeln, die angeben, was die Adaption bewirken soll.

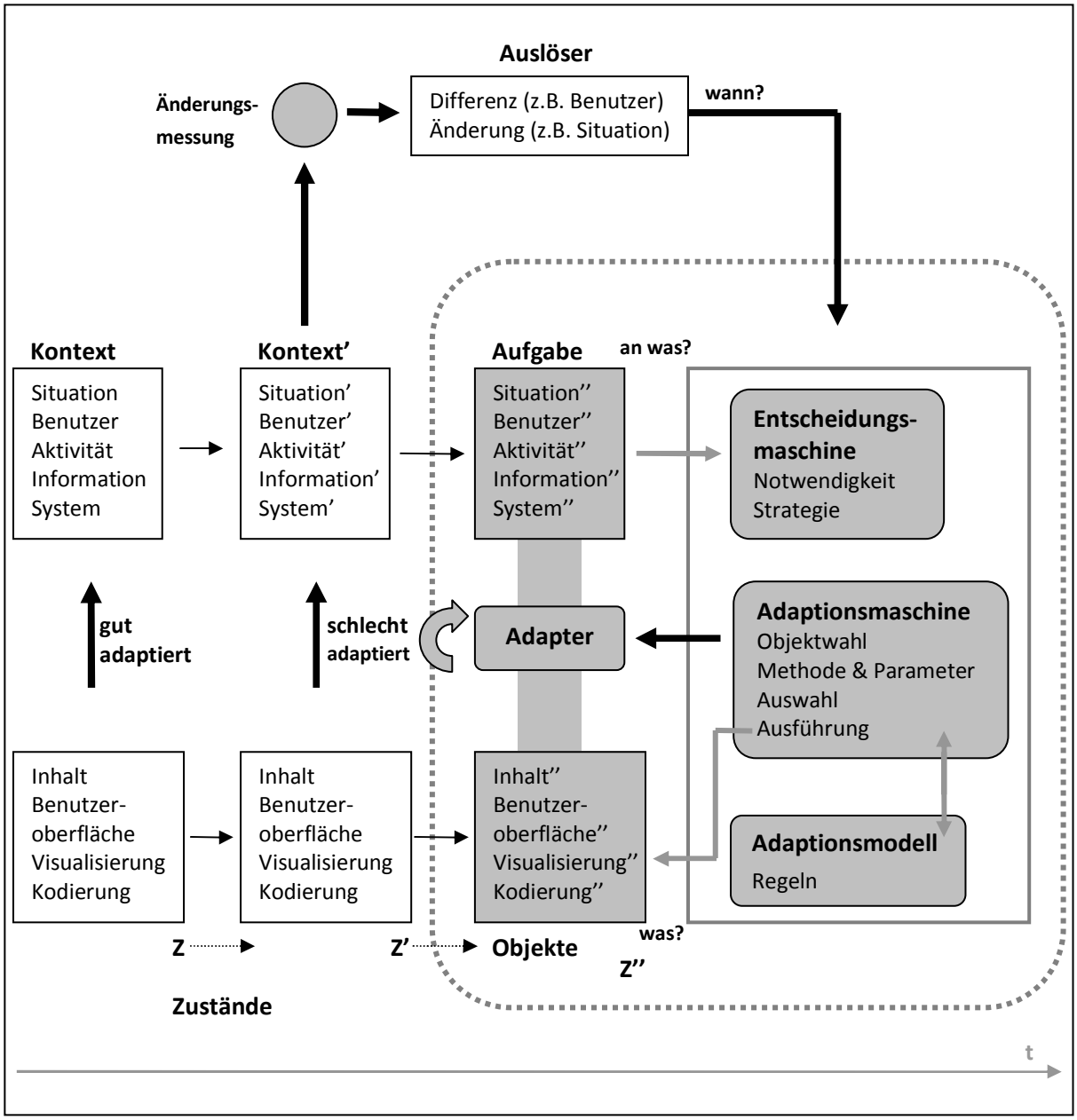


Abb. 3.5: Der Adaptionsprozess der mobilen Kartographie (nach REICHENBACHER, 2003)

3.2.5 Egozentrische Karten

In der Kartographie hat sich der Begriff der egozentrischen Karte für eine personalisierte Karte etabliert. Eine egozentrische Karte besitzt folgende Eigenschaften (MENG, 2004):

- Sie ist adaptiv, speziell wenn das Egozentrum einen flüchtigen Charakter hat.
- Sie dient sehr effizient kurzfristigen, speziellen und/oder zeitkritischen Aufgabenstellungen, besonders in einem mobilen Benutzungskontext.
- Sie bietet eine *one-to-one*- statt *one-fit-for-all*-Lösung, da sie auf die speziellen Benutzerbedürfnisse zugeschnitten ist.

- Das Egozentrum und die restlichen Inhalte sind differenziert dargestellt, dadurch ist sie graphisch auffällig und inhaltlich leicht zugänglich.

Die vorangegangen, zunächst theoretischen Überlegungen der vorherigen Kapitel bezüglich der Adaption einer mobilen Karte werden mit dem Begriff der egozentrischen Karte zunehmend praktischer, denn für solch eine schlägt MENG (2005) mehrere Designmethoden vor, die eine Adaption im mobilen Nutzungskontext umsetzen. Diese sind u.a. (MENG, 2005):

- **Zentrierung:** Zentrierung der Karte auf den Benutzerstandort.
- **Verschachtelte *Level-of-Detail* (LoD):** Ein höheres LoD in der unmittelbaren Umgebung des Benutzers bzw. entlang seiner Route ist eingebettet in ein niedrigeres LoD der Gebietsübersicht.
- **standortabhängige LoD**: Wenn der Fokus des Benutzers auf einem oder mehreren Standorten liegt, werden die Objekte in unmittelbarer Nähe dieser Standorte in einem höheren größeren Maßstab, der mehr Details enthält, dargestellt als weiter entfernt liegende. Dieser Maßstabsübergang wird durch verschiedene Verzerrungstechniken realisiert.
- **Beweglicher Rahmen:** Bei der vorhergehend beschriebenen Methode besteht der Nachteil, dass auf Grund der Verzerrungen Entfernungen nur schlecht geschätzt werden können. Meist braucht der Benutzer nur zwei LoD: eine Straßenkarte für die unmittelbare Umgebung und eine Übersichtskarte des Gesamtgebietes. Dies kann durch einen Rahmen realisiert werden, der das feinere LoD anzeigt und über der Übersichtskarte beliebig bewegt werden kann.
- **Raumkontraktionen:** Wenn der Benutzer die Visualisierung mehrerer weit auseinander liegender Orte benötigt, wird die Gesamtübersicht kontrahiert, um alle Inhalte zugleich auf dem Bildschirm sichtbar zu machen. Das LoD und die topographischen Relationen bleiben unberührt.
- **Einzelfenster mit Details auf Anfrage:** Die begrenzte Bildschirmgröße mobiler Endgeräte lässt eine Darstellung in mehreren Fenstern nebeneinander nicht zu. Deshalb bietet es sich an, Hintergrundinformationen zu verbergen und nur auf Anfrage sichtbar zu machen. Zum Beispiel ruft der Klick auf einen POI ein zusätzliches Fenster mit Detailinformationen auf.

3.3 User Generated Content in der Kartographie

Vor Beginn des 21. Jahrhunderts war es eine allgemein gültige Annahme, dass man sowohl einen Universitätsabschluss benötigt, um die Erde zu vermessen und die dabei gewonnen Informationen auf Papier oder in den Computer zu transkribieren, als auch eine teure Ausrüstung. Das hat sich drastisch geändert, als US-Präsident Bill Clinton im Jahr 2000 die Entfernung der *Selective Availability* im GPS-Signals verkündete und damit eine verbesserte Genauigkeit für preiswerte GPS-Empfänger

gewährleistete. Die Genauigkeit der Positionsbestimmung wurde von 100 Meter auf 10 bis 15 Metern verbessert und zugleich die Grundlage gelegt für eine Reihe von Projekten, die auf benutzergenerierten Kartierungen basieren wie zum Beispiel *OpenStreetMap* (HAKLAY/WEBER, 2008). *OpenStreetMap* wurde im Jahr 2004 gegründet und hat seit 2007 ein so rasantes Wachstum zu verzeichnen, dass innerhalb von europäischen Ballungszentren eine sehr gute Abdeckung gegeben ist. Neben *OpenStreetMap* sind *Wikipedia*, *YouTube* oder *Flickr* weitere Indikatoren für die große Bedeutung von *User Generated Content* im Internet, auch außerhalb der Kartographie (ZIPF, 2009).

3.3.1 Vorteile, Nachteile und Kritik

Beschäftigt man sich eingehender mit *User Generated Content*, stößt man unweigerlich auf die Frage der Verlässlichkeit dieser Daten und auf damit verbundene Kritik und ein Abwägen von Vor- und Nachteilen, was an dieser Stelle kurz wiedergegeben werden soll.

Ein klarer Vorteil von nutzergenerierten Inhalten sind die geringen Kosten. Zwar sind am Anfang Investitionen vonnöten, um Benutzer für das Projekt begeistern und so Daten sammeln zu können, aber da die Datennutzer meist gleichzeitig die Autoren sind, welche sich mit dem Projekt identifizieren und somit auch nicht ohne Weiteres zur Konkurrenz wechseln, sorgen sie dafür, dass dieses bekannt wird. Ein weiterer Vorteil, der auch für viele Endanwender das ausschlaggebende Argument zu sein scheint, ist die hohe Aktualität. Da viele Benutzer gleichzeitig die Realität beobachten, werden Änderungen schneller be- und auch vermerkt.

Als Negativum wird oft der Nachteil genannt, dass Benutzer nur das in eine Karte eintragen, was sie selber interessiert. Des Weiteren ist eine Qualitätssicherung in Form einer Kontrolle der Daten seitens des Betreibers nicht immer möglich. Oftmals kontrollieren sich die Benutzer gegenseitig. Die größte Angst bei *User Generated Content* ist jedoch Vandalismus. Benutzer befürchten eine Manipulation, Verfälschung oder gar Löschung ihrer Daten. Ein solcher Fall ereignete sich bei *OpenStreetMap* im Jahr 2007, stellt jedoch eine seltene Ausnahme dar: ein Erotikgeschäft wurde zur Hauptstadt ‚erhoben', was dessen Namen sehr populär machte (POLCHAU, 2009). An dieser Stelle spielt auch die Größe der Community eine Rolle, denn umso mehr Mitglieder die Inhalte betrachten, desto schneller und wahrscheinlicher fallen Fehler auf und können behoben werden, womit eine bessere und verlässlichere Qualitätssicherung gegeben ist sowie eine stetig wachsende Informationsdichte und ein sich zunehmend verbessernder Informationsgehalt. Dies wird auch als *CrowdSourcing* bezeichnet (ZIPF, 2009).

Die meiste Kritik erntet *User Generated Content* in Bezug auf die vermutete mangelnde Qualität. Diese wird vorwiegend in der fehlenden Redaktion, den anonymen Veröffentlichungen und der zweifelhaften Vertrauenswürdigkeit der Autoren gesehen. Professionelle Anbieter befürchten einen Verlust ihres Publikums und einen Preissturz, der möglicherweise nötig ist, um ersterem entgegen zu wirken. Dabei ist fraglich, ob professionelle Inhalte überhaupt mit nutzergenerierten Inhalten gleichgesetzt werden können, denn da es sich um ein freiwilliges Sammeln und Veröffentlichen von Daten handelt, haben solche Projekte auch keinen professionellen Anspruch, sondern wollen durch nichtkommerzielle Unverfälschtheit und Quantität bestechen (WORMS, 2008).

Der eben erwähnten zweifelhaften Vertrauenswürdigkeit von Autoren wird in Form eines Mechanismus Rechnung getragen, bei dem Benutzer andere Benutzer für alle sichtbar bewerten, wie beispielsweise bei *eBay*. Ähnlich wird bei *Wikipedia* vorgegangen, dessen fragwürdige Inhalte als solche gekennzeichnet werden können bzw. bei *Facebook*, wo Benutzer ein unangebrachtes Verhalten anderer User öffentlich machen können. Ein ähnlicher Mechanismus existiert jedoch noch nicht für kartographische nutzergenerierte Inhalte (GARTNER/SCHMIDT, 2010).

3.3.2 Motivationen für Nutzung und Erstellung von User Generated Content

Gründe dafür, kartographische Inhalte von unprofessionellen Erzeugern zu nutzen, liegen vor allem darin, dass Webseiten wie *OpenStreetMap* kostengünstige Datenquellen sind - teilweise sogar die einzigen; speziell in Gebieten, in denen der Zugang zu geographischen Informationen eine Frage der nationalen Sicherheit ist (GOODCHILD, 2007a).

Die Motivation, aus welcher heraus Benutzer selbsterhobene geographische Informationen der Allgemeinheit zur Verfügung stellen, kann verschiedener Art sein (COLEMAN/GEORGIADOU/LABONTE, 2009; STÖCKL/GRAU/HESS, 2006):

- Altruismus
- professionelles oder persönliches Interesse
- intellektuelle Stimulation
- Sicherung oder Verbesserung einer persönlichen Investition
- soziale Anerkennung, Wertschätzung
- verbesserter Ruf
- Anerkennung eines Expertenstatus für ein bestimmtes Gebiet
- Möglichkeit der kreativen und unabhängigen Selbstverwirklichung/-darstellung
- Vorrangstellung
- Gruppenzugehörigkeit
- einfache Veröffentlichung eigener Produktionen

Daneben existieren ebenso negative, aber nicht weniger wichtige Motivationen, die nicht vernachlässigt werden sollten. Nicht alle Produzenten von nutzergenerierten Inhalten sind an einer Bereitstellung objektiver und verlässlicher Informationen interessiert. Diese Motivatoren sind leicht zu identifizieren (COLEMAN/GEORGIADOU/LABONTE, 2009):

- Schaden anrichten
- Verbreitung falscher Informationen

- böswillige oder kriminelle Absichten

3.3.3 EveryTrail - ein Beispiel für User Generated Content

An dieser Stelle soll nicht *OpenStreetMap*, das Paradebeispiel für nutzergenerierte Inhalte in der Kartographie aufgeführt werden, sondern die Plattform *EveryTrail*, die seit Juli 2007 existiert, jedoch mehr den touristischen Bereich bedient.

Mit *EveryTrail* ist es möglich, Reisen zu planen und Reiseerfahrungen auf interaktiven Karten gespickt mit Fotos oder Videos entlang eines *Trails* zu teilen. Davon machen bereits mehr als eine Million von Benutzern Gebrauch mit einer jährlichen Zuwachsrate von 500% (GPS BUSINESS NEWS, 2009). Täglich werden etwa 1.500 neue nutzergenerierte *Trails* veröffentlicht.

Hinter *EveryTrail* steht die Firma *GlobalMotion*, eine Firma mit Sitz in Kalifornien, die sich auf die Entwicklung von touristischen Anwendungen spezialisiert hat.

Die Teilnahme bei *EveryTrail* ist kostenlos. Über 300.000 *Trails* in über 140 Ländern (GLOBALMOTION, 2010) können eingesehen werden, welche kategorisiert sind in Road-Trips, Sightseeing-Touren, Segel-, Wander- und Radtouren, Fliegen, Drachenfliegen, Geocaching, Skifahren, Kajak-Trips und mehr. Es gibt mobile Anwendungen für iPhone, BlackBerry, Android und Windows Mobile, die auf *Trails* von *EveryTrail* zurückgreifen bzw. mit denen es möglich ist, *Trails* selber zu erstellen und bei *EveryTrail* zu veröffentlichen. Eine davon wird in Kapitel 4.1 untersucht.

Benutzer können *Trails* erstellen, indem sie diese mit einem GPS-Gerät oder einem Handy, das über eine GPS-Funktion und gegebenenfalls eine *EveryTrail*-Anwendung verfügt, aufzeichnen. Die dabei entstandene GPX-Datei und die auf der Tour aufgenommenen Fotos können anschließend bei *EveryTrail* hochgeladen und in einer interaktiven Karte von *Google Maps* visualisiert werden. Liegt keine GPX-Datei vor, kann die Tour auch direkt in die Karte eingezeichnet werden. Die Fotos können wahlweise automatisch anhand ihres Zeitstempels ihren Aufnahmeorten zugeordnet werden oder manuell. Wurden der *Trail* und die Fotos über ein Handy mit einer *EveryTrail*-Applikation aufgenommen, wird die automatische Zuordnung bereits innerhalb der Applikation vorgenommen. Abschließend wird der *Trail* veröffentlicht und ist somit zugänglich für die gesamte *EveryTrail*-Community. Auch nichtangemeldete Benutzer können Touren im Internet einsehen, jedoch ist ein Download nicht möglich. Gegen eine geringe Gebühr können registrierte User *Trails* aufs Handy herunterladen inklusive einer detaillierten Karte, Fotos, einem Höhenprofil der Tour, Beschreibungen und *Points of Interest* (EVERYTRAIL).

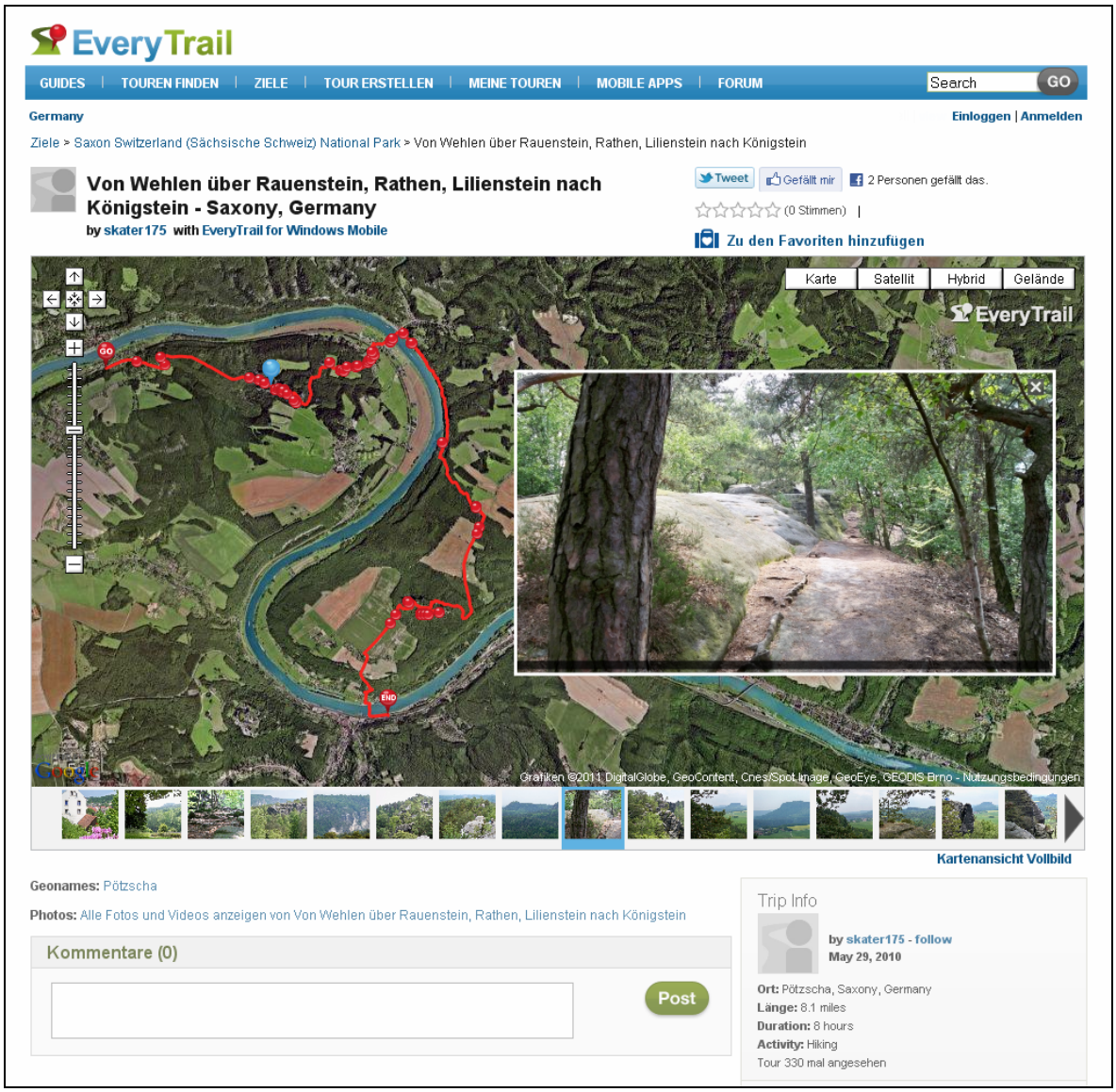


Abb. 3.6: Ansicht einer Tour bei EveryTrail (EVERYTRAIL)

3.4 Kartographische Kommunikationsmodelle

In den 70er Jahren des 20. Jahrhunderts wurde der Kommunikationsbegriff erstmals nachhaltig in der Kartographie untersucht, auch im Zusammenhang mit kybernetischen, zeichen- und informationstheoretischen Aspekten der Kartenherstellung und Kartennutzung. Der Begriff der kartographischen Information im Rahmen eines Sender-Kanal-Empfänger-Modells wurde geprägt, basierend auf der zu dieser Zeit neuen Erkenntnis, dass die Funktionalität von Karten nur gewährleistet werden kann, wenn bei der Konzeption ihre Nutzung berücksichtigt wird. Des Weiteren entstanden Arbeitsbereiche in der Kartographie, die theoretisch und methodisch geprägt sind und deren Aufgabe bis heute in der zeichentheoretischen Untersuchung und Beschreibung einzelner Komponenten des kartographischen Kommunikationsprozesses besteht. In diesem Zusammenhang wurde jener als eine kartographische

Kommunikationskette beschrieben und als ein System von Elementen der kartographischen Informationsübermittlung betrachtet, dessen Erfolg in der kartographischen Kommunikation von bestimmenden (‚determinierenden') Faktoren gesteuert wird (BOLLMANN/KOCH, 2001).

3.4.1 Die kartographische Kommunikationstheorie

Allgemein betrachtet, lässt sich ein Kommunikationsprozess beschreiben mit „Wer sagt was zu wem und mit welcher Wirkung?". Ein solcher Vorgang findet zwischen Kommunikatoren bzw. Kommunikationsgrößen (Menschen, Automaten, Tieren) statt und wird unterschieden in (HAKE/GRÜNREICH, 1994):

- **Duplex-Kommunikation:** Hierbei handelt es sich um eine dialogisierende Kommunikation, d.h. eine wechselseitige Beziehung. In der Kartographie findet eine Duplex-Kommunikation vor allem in Form von Benutzeranfragen, graphischen Manipulationen bzw. Adaption interaktiver Karten statt.
- **Simplex-Kommunikation:** Bei dieser diagnostischen Kommunikation findet eine einseitige Erfassung, Beobachtung, Analyse und Erkenntnis der Außenwelt statt. In der Kartographie ist dies die Kommunikation mit klassischen Papierkarten und nichtmanipulierbaren Datenbeständen, also *view-only-Karten*.

Die Kommunikation dient der Informationsübertragung und in ihrer Wirkung übt sie mit der empfangenen Information einen Einfluss auf den Kommunikator aus. Eine Information stellt hier eine Nachricht oder eine Mitteilung dar.

Innerhalb der kartographischen Kommunikationstheorie werden zwei Aspekte unterschieden: die Informations- und die Zeichentheorie (HAKE/GRÜNREICH, 1994).

Die Informationstheorie ist eine reine, mathematisch fundierte Syntax, in der der Sinngehalt der zu übermittelnden Nachricht unberücksichtigt bleibt. Sie beschreibt den Vorgang der einseitigen Informationsübermittlung mit einem Schema, dessen Begriffe weitestgehend aus der Nachrichtentechnik stammen und welches in Abbildung 3.7 dargestellt ist. Demnach wird der Inhalt einer Information beim Sender als ein Zeichen codiert und über einen Kanal an den Empfänger übermittelt, von welchem sie decodiert wird. Auf den Informationskanal können jedoch Störeinflüsse einwirken, die die Zeichenbildung und somit auch den Inhalt der Nachricht beeinflussen und verändern. Die Informationen werden stets in codierter Form als Zeichen übermittelt, wobei diese Zeichen oder Zeichenfolgen die Realisierung von Informationsinhalten darstellen (HAKE/GRÜNREICH/MENG, 2002).

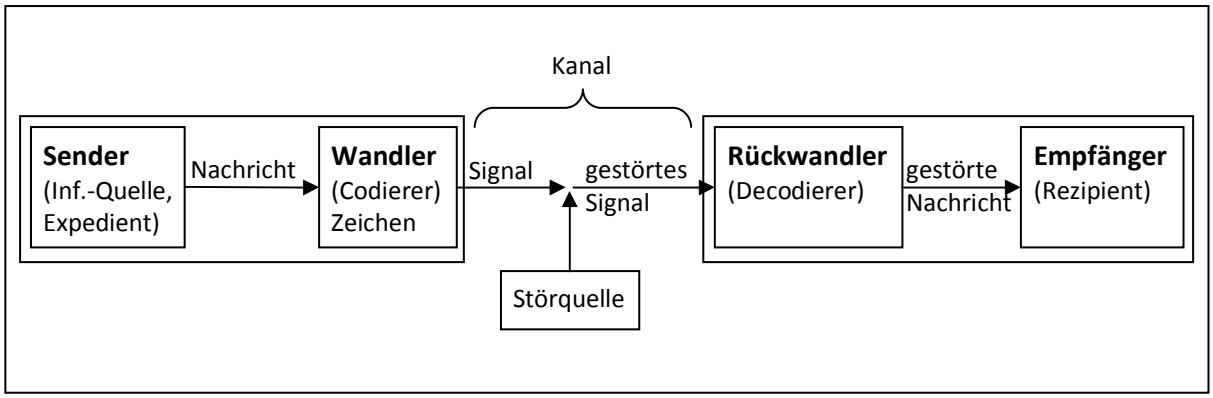


Abb. 3.7: Schema der Informationsübertragung in Form eines Sender-Kanal-Empfänger-Modells (nach HAKE/GRÜNREICH, 1994)

Eine Duplex-Kommunikation ist nur dann sinnvoll, wenn bei den beteiligten Kommunikatoren ein bestimmtes gemeinsames Repertoire an Zeichen und Zeichenbedeutungen, d.h. ein gemeinsamer Zeichenvorrat, vorhanden ist. Die Zeichentheorie (Semiotik) beschäftigt sich mit der Frage des Sinngehaltes und der Wirkung dieser Zeichen. Sie ist also die Erkenntnistheorie der Zeichen und unterscheidet drei sogenannte Zeichendimensionen (HAKE/GRÜNREICH/MENG, 2002):

- **syntaktische Dimension**: Sie regelt die formale Bildung der Zeichen und ihre Beziehungen untereinander.
- **semantische Dimension:** Sie bringt die sogenannte Zeichenbedeutung zum Ausdruck, betrifft also die Beziehung der Zeichen zu den Objekten, die sie darstellen sollen.
- **pragmatische Dimension:** Sie regelt die Beziehung zum wahrnehmenden Subjekt und nimmt damit Einfluss auf dessen Verhaltenweise.

Der kartographische Kommunikationsprozess wurde in zahlreichen Modellen dargestellt und zusammengefasst, welche im Folgenden vorgestellt werden sollen.

3.4.2 Bestehende kartographische Kommunikationsmodelle

Bei den in diesem Kapitel beschriebenen kartographischen Kommunikationsmodellen ist in annähernd jedem Modell die ‚Urform' des Sender-Kanal-Empfänger-Modells erkennbar. Dieses Modell ist aus der mathematischen Theorie der Kommunikation der beiden Mathematiker SHANNON und WEAVER (1949) hervorgegangen.

Ein Kommunikationsmodell kann auf zweierlei Weise ausgerichtet sein. Definiert es die Phasen, aus denen ein Kommunikationsvorgang aufgebaut ist, handelt es sich um ein Prozessmodell. Ein Systemmodell stellt es hingegen dar, wenn es die Komponenten definiert, die an diesem Vorgang

beteiligt sind (HEIDMANN, 1999), wobei eine klare Abgrenzung dieser zwei Arten nicht immer möglich ist.

Auch in dem ersten bedeutenden kartographischen Kommunikationsmodell von KOLÁČNÝ aus dem Jahre 1969 (siehe Anhang I) ist das Sender-Kanal-Empfänger-Modell gut erkennbar: der Kartograph fungiert als Sender, die Karte stellt den Kanal dar und der Kartenbenutzer den Empfänger. Als vierte Hauptkomponente kommt das Universum des Kartographen und Kartenbenutzers hinzu, aus dem Informationen für die Codierung entnommen werden. Die Wechselbeziehung zwischen Kartograph und Kartenbenutzer wird anhand eines zirkulären Kommunikationsmodells in Dreiecksform dargestellt, das Begriffe aus der Informations-, Kommunikations- und anderer Sozialtheorien enthält (FREITAG, 1992). Kartograph und Kartenbenutzer verfügen über einen gemeinsamen Zeichenvorrat und werden von zahlreichen determinierenden Faktoren beeinflusst wie ihren Fähigkeiten oder psychischen Prozesse (BOLLMANN/KOCH, 2001).

Nach jenen Faktoren hat OGRISSEK sein Kommunikationsmodell aus dem Jahre 1974 benannt (siehe Anhang II). In dem einseitig ausgerichteten Modell wird deutlich, dass die Wirkungszusammenhänge der einzelnen Glieder der kartographischen Kommunikationskette maßgeblich sind für die Effektivität des kartographischen Kommunikationsprozesses. Die Gesamtmenge der möglichen Informationen, die zur Befriedigung des Informationsbedürfnisses zur Verfügung steht, ist in zwei Teilmengen gegliedert: in verbale und kartographische Informationen. Die Informationsaufbereitung und -formulierung wird von verschiedenen determinierenden Faktoren beeinflusst, die kartenredaktioneller und kartengestalterischer Natur sind (OGRISSEK, 1987).

BREETZ lieferte 1982 in der DDR einen neuen Ansatz zur Strukturierung der kartographischen Kommunikationskette in Form eines Prozessmodells (siehe Anhang III). Es handelt sich um eine Weiterentwicklung des Grundschemas von OGRISSEK (1974). Die Kartennutzung als ein gleichrangiges Glied der Kommunikationskette wird ebenfalls differenziert dargestellt. In diesem Modell werden determinierende Faktoren nicht explizit ausgewiesen (OGRISSEK, 1987).

Ein anderer Ansatz kam 1983 von PRELL (siehe Anhang IV). Sein Modell zeigt kartographische bzw. topographische Tätigkeiten sowie den Kartenleseprozess als Ablaufschema und darin integriert die kommunikative Zeichensituation, wie sie durch die Herstellung und Benutzung einer topographischen Karte als visuellem Kommunikationsmittel entsteht. Die Darstellung basiert auf theoretischen Untersuchungsergebnissen von RATAJSKI (1977) zu Informationsverlust und Informationsgewinn durch die kartographische Kommunikation und integriert diesen Ansatz mit eigenen Erweiterungen und Abweichungen in das Modell (PRELL, 1983). Es findet eine genaue Betrachtung der Bewusstseinsinhalte von Karten und Kartennutzer und deren Überlappungsbereichen statt, wobei zwischen fehlerfreien, fehlerbehafteten, redundanten, vom Benutzer nicht verstandenen und indirekten Informationen unterschieden wird. Auch in diesem Modell ist das Sender-Kanal-Empfänger-Modell wieder deutlich erkennbar.

Ebenfalls PETERSON adaptierte 1995 das Modell von KOLÁČNÝ (1969), lässt aber die determinierenden Faktoren außer Acht (siehe Anhang V). Der Ausgangspunkt seines Modells ist, dass der Kartograph neben der Gestaltung einer Karte auch generelle Kartennutzungsbedingungen in Form eines User Interfaces konzipiert, wodurch dem Kartennutzer mehr Kontrolle über den Kommunikationsprozess

gegeben wird und er über die Grenzen einer statischen Karte hinausgehen kann (PETERSON, 1995). Das Ergebnis der Abstraktion der Realität des Kartographen ist demnach eine interaktive Karte. Der Kartenbenutzer nimmt diese Karte wahr, interpretiert sie, greift auf etwaige Funktionalitäten zu und gestaltet die Karte. Letzteres wird im Modell durch eine sogenannte *Feedback Loop* dargestellt.

In ihrem Modell von 2002 zeigen HAKE, GRÜNREICH und MENG die Bildung von Primär- (Fachmann), Sekundär- (Kartograph) und Tertiärmodell (Benutzer) auf und geben als Sekundärmodelle auch Geoinformationssysteme, digitale Karten und multimediale Darstellungen an (siehe Anhang VI). Da ebenfalls der Dialog des Benutzers mit dem Kartographen oder Fachmann bzw. der Dialog des Kartographen mit dem Fachmann eingezeichnet ist, erweitert sich die Informationskette zu einem oder mehreren Regelkreisen, wenn die kartographische Wiedergabe zu Vergleichen genutzt wird (HAKE/GRÜNREICH/MENG, 2002).

In einem recht komplexen Modell aus dem Jahr 2000, in welches wiederholt determinierende Faktoren einfließen, wird erstmals die mögliche Datenintegration durch den Benutzer in ein IS oder KIS aufgezeigt (siehe Anhang VII). Dieses Modell stammt von LECHTHALER. Ähnlich in einem Modell von KELNHOFER von 2003 (siehe Anhang VIII). Hier steht im Zentrum des Modells ein *Tool Set*, mithilfe dessen der Kartennutzer selbst zum ‚Kartenmacher' wird. Gleichzeitig wird seine notwendige geographische und erforderliche kartographische Kompetenz in Frage gestellt. KELNHOFER (2003) bezeichnet den Kartographen aktuell sehr zutreffend als ‚Werkzeugbereitsteller', der für Basistopographien und die Datenvorbereitung verantwortlich ist.

In Anhang IX sind neben allen eben vorgestellten kartographischen Kommunikationsmodellen weitere aufgelistet mit Autor, Titel, Entstehungsjahr und weiteren Eigenschaften.

3.4.3 Ableitung eines Kommunikationsmodells für mobile, interaktive Karten

Zwar werden in den neueren der im letzten Kapitel beschriebenen kartographischen Kommunikationsmodelle interaktive Karten bzw. teilweise auch der Benutzer als ‚Kartenmacher' berücksichtigt, jedoch nicht die Adaption einer Karte an einen Benutzer und seinen Nutzungskontext, obwohl speziell einige determinierende Faktoren aus KOLÁČNÝ's Modell (1969), die auf den Kartenbenutzer einwirken, den Aspekten des Kontext entsprechen bzw. nahe kommen: Bedarf, Interessen, Ziele, Kenntnisse, Erfahrung, Fähigkeiten, psychische Prozesse, äußere Bedingungen. Einzig KELNHOFER (2003) führt in seinem Modell eine ‚Nutzerdefinierte thematische Karte' auf.

Die Adaption muss also in einem neuen, abgeleiteten Modell Berücksichtigung finden ebenso wie die schon erwähnte Tatsache, dass der Kartennutzer im Kommunikationsprozess auch die Rolle des Kartographen übernimmt. Letzterer rückt damit mehr in den Hintergrund und stellt sein Fachwissen in Form von Komponenten wie Basiskarten, Software oder Interaktionsmöglichkeiten zur Verfügung, die der Kartennutzer über Benutzerschnittstellen verwenden kann, um seine eigenen Daten einzubinden (HOFFMANN, 2010).

Es kann also nicht nur von einem großen ‚Datentopf', in den sowohl Kartograph als auch Kartenbenutzer Informationen eingeben, ausgegangen werden, sondern zusätzlich von einer Anzahl an

Interaktionsmöglichkeiten, die der Kartograph dem Benutzer bereitstellt. Aus dieser Gesamtmenge wird - abhängig vom Benutzer und seinem Nutzungskontext - eine Auswahl getroffen, die ihm auf seinem mobilen Endgerät anschließend präsentiert wird. Somit wären die Ziele der mobilen Kartographie in das Modell eingeflossen: Reduktion von Informationen und Interaktionen auf das benötigte Maß.

Das für mobile, interaktive Karten abgeleitete Kommunikationsmodell ist in Anhang X zu finden. Diesem wurde das Sender-Kanal-Empfänger-Modell zu Grunde gelegt. Auf der linken Seite steht der Kartograph als Sender und Kodierer. Er lässt sich aufspalten in Gestalter und Entwickler. Der Entwickler liefert Algorithmen, Software und Interaktionsmöglichkeiten für den Benutzer, der Kartograph Basiskarten und *Style-Files*, die der Gestaltung bzw. Kodierung von Informationen aus der realen Welt dienen. Hierbei wird der Kartograph und somit auch das Ergebnis des Kodier- und Sendevorgangs vor allem von seinen kartographisch-gestalterischen und informatisch-technischen Fähigkeiten und Erfahrungen beeinflusst. Diese bilden also den Kontext (im Modell in hellgrauer Schrift) des Kartographen, welcher ja nicht nur bei der Kartennutzung, sondern ebenso bei der Kartenerstellung eine Rolle spielt. Nach KOLÁČNÝ (1969) wären dies die determinierenden Faktoren.

Auf der rechten Seite steht der Benutzer, der hier als *ProdUser* bezeichnet wird. Der Ausdruck *ProdUser* ist ein Wortspiel, das die Begriffe *Producer* und *User* vereint. Gemeint ist hiermit, dass der Benutzer, also der *User*, nicht nur Konsument, sondern auch Produzent, d.h. *Producer* ist. Ein ähnlicher aufkommender Begriff ist der des *Prosumers*. Der Ausdruck *ProdUser* passt somit sehr gut in ein Kommunikationsmodell der mobilen Kartographie, da ebenfalls der Benutzer aufgesplittet werden kann, nämlich in Datenerzeuger und Datennutzer. Der *ProdUser* fungiert demzufolge einerseits als Sender, andererseits auch als Empfänger und Dekodierer. Als Sender kann er Rohdaten in die vom Kartographen zur Verfügung gestellte Basiskarte einbinden und kann sie mit qualitativen Informationen und Fotos versehen. Die Kodierung der durch den Benutzer gelieferten Rohdaten geschieht durch die vom Kartographen erstellten *Style-Files*. Zwar wird der Benutzer hier sehr treffend als *ProdUser* bezeichnet, jedoch muss Datennutzer und Datenerzeuger nicht zwingend in derselben Person vereint sein. Daher wäre im Modell auch die Schreibweise *(Prod)User* denkbar. Aus dieser Schreibweise geht hervor, dass der *ProdUser* sehr unwahrscheinlich nur Datenerzeuger ist, sondern auch gleichzeitig Datennutzer, wohingegen ein reiner Datennutzer realistisch ist. Die auf den Benutzer als Datenerzeuger wirkenden Einflüsse, die seinen Kontext bilden, sind seine Erfahrungen, Fähigkeiten, Ziele, Präferenzen und Interessen. Letzteres bestimmt vor allem, worüber er Inhalte erstellt. Durch eben diese Art der Inhalte unterscheidet er sich vom Kartographen als Sender, genauso wie durch seine Erfahrungen und Fähigkeiten, die sich in der Qualität seiner erzeugten Inhalte widerspiegelt. Die Erfahrungen und Fähigkeiten des Kartographen garantieren eine höhere kartographische Güte, wobei zwischen verschiedenen Kartographen auf Grund von vergleichbaren Ausbildungen Unterschiede in Fähigkeiten und Erfahrungen weniger dominant sind als in der heterogenen Masse der *ProdUser*.

Den Mittelpunkt des Modells bildet die mobile Anwendung. Diese setzt sich zusammen aus Hardware und Software. Die Hardware stellt das mobile Endgerät dar, dessen Eigenschaften ebenfalls den Kontext des Benutzers ausmachen, jedoch auch vom Kartographen bei der Erstellung der Basiskarten, *Style-Files*, Algorithmen und Interaktionen beachtet werden müssen. Die Software wird einerseits

gebildet aus den von Kartographen und Benutzer zusammengetragenen thematischen Inhalten Karte, Text und Bild. Nach WOLODTSCHENKO (2010) sind dies die semiotischen Metavariablen. Andererseits besteht die Software aus den vom Kartographen entwickelten Interaktionsmöglichkeiten. Aus diesen Bestandteilen wird dem Kontext des Datennutzers entsprechend eine Auswahl getroffen und auf dem mobilen Endgerät, der Hardware, visualisiert. Damit wurden die Inhalte adaptiert.

Anhand der drei Komponenten Kartograph, mobile Anwendung und *ProdUser* wurde das Sender-Kanal-Empfänger-Modell aufgegriffen, wobei jedoch zwei Sender auftreten: der Kartograph und der *ProdUser* als Datenerzeuger. Dadurch ist die ursprüngliche Kettenform nicht mehr gegeben. Als Kanal für die Kommunikation dient die mobile Anwendung.

Über dem abgewandelten Sender-Kanal-Empfänger-Modell ist die reale Welt platziert, aus welcher Informationen entnommen werden. Mit diesem vierten Element ist wieder die klassische Dreiecksform, wie sie auch von KOLÁČNÝ (1969) verwendet wurde, erreicht. Die reale Welt enthält zwei Teilmengen, welche sich überschneiden: die Welt des Kartographen und die Welt des *ProdUsers*. Diese Überschneidung stellt einerseits den gemeinsamen Zeichenvorrat dar, der vorhanden sein muss, um eine Kommunikation mit einer Karte als Medium überhaupt möglich zu machen; andererseits muss auch eine Überschneidung vorliegen, damit der *ProdUser* fähig ist, mit den vom Kartographen bereitgestellten Interaktionsmöglichkeiten und der Software umzugehen und mit Hilfe derer eigene Daten einzubinden sowie seinen Kartenleseprozess erfolgreich stattfinden lassen zu können. Auch dieser realen Welt entspringen Einflüsse, die in Form von äußeren Bedingungen sowohl auf den Kartographen als auch auf den *ProdUser* einwirken und wiederum deren Fähigkeiten, Interessen, Ziele etc. beeinflussen und infolgedessen den Kontext mit formen. Die Entnahme von primären Geoinformationen aus der realen Welt durch den Kartographen erfolgt durch Vermessung, Kartierung, Messung und Zählung; der *ProdUser* als Datenerzeuger geht dabei eher einer Aufzeichnung der realen Welt durch ein GPS-Gerät oder einen Fotoapparat, durch Zählung und Beobachtung nach. Diese Informationen in Form von abgeleiteten Geoinformationen beeinflussen die Welt des *ProdUsers* nach dem Vorgang der Karteninterpretation, indem sie Handlungen zur Folge haben.

4. Touristische Applikationen für mobile Endgeräte

Applikationen für mobile Endgeräte, speziell für Mobiltelefone, gibt es schon länger. 2008 war Apple der erste Hersteller, der mit seinem *App-Store* ein gut funktionierendes Geschäftmodell entwickelte. Seitdem haben sich weitere Verkaufsportale etabliert: der *Android Marketplace* für Mobiltelefone mit einem Android-Betriebssystem, die *App Worl*d für BlackBerry-Handys, der *Window Marketplace for Mobile* von Microsoft oder der *Ovi Store* von Nokia (WIKITUDE, 2010).

Wie Abbildung 4.1 zeigt, sind der *Android Marketplace* und der *Window Phone 7 Marketplace* in den USA im Januar 2011 die Stores mit den höchsten Zuwachsraten an Applikationen im Vergleich zum Vormonat. Ferner ist der *Android Marketplace* der einzige Store, in dem mehr kostenfreie als kostenpflichtige Apps zum Download zur Verfügung stehen (63%) (DISTIMO, 2011).

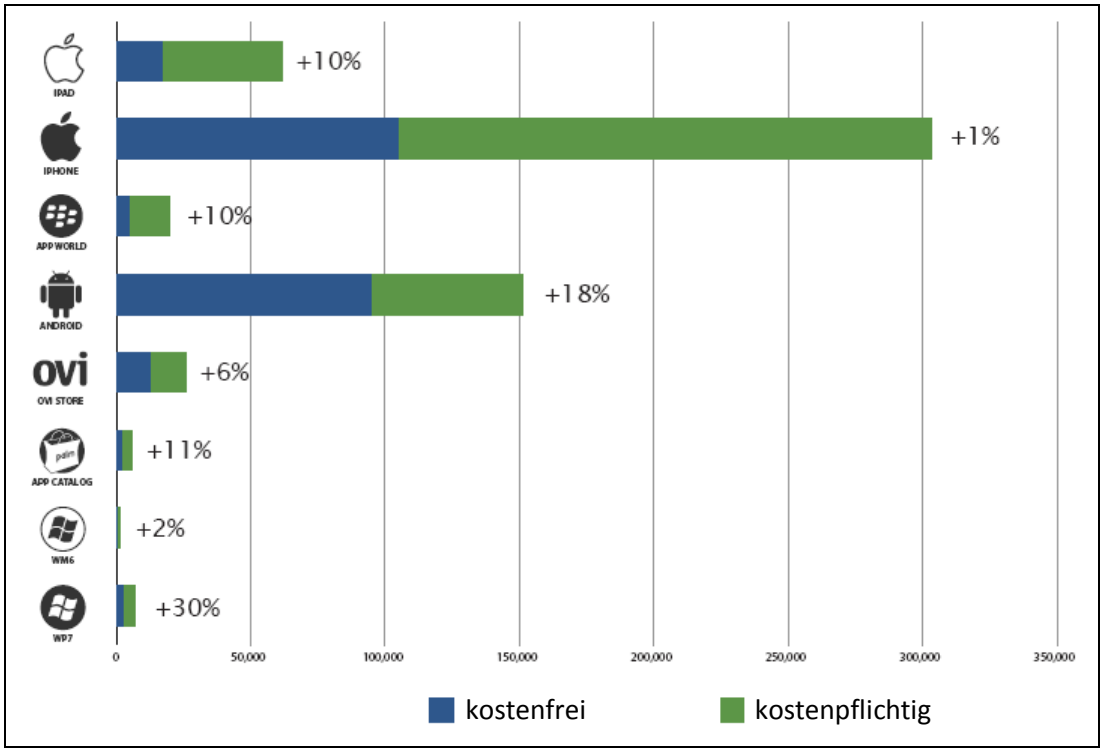


Abb. 4.1: Anzahl der verfügbaren mobilen Applikationen im Januar 2011 in den USA (DISTIMO, 2011)

Mobile Applikationen können in verschiedene Kategorien eingeteilt werden, die in Tabelle 4.1 aufgeführt sind.

Kommunikation	**Spiele**
- Email - Mobile Web und Internet-browser - Newsreader - On Device Portal - Soziale Netzwerke	- Puzzle- und Strategiespiele - Karten- und Casinospiele - Action- und Abenteuerspiele - Sportspiele
Multimedia	**Organizer**
- Bildbetrachter - Präsentationsbetrachter - Video-Wiedergabe - Audio-Wiedergabe - Streaming-Player	- Kalender - Taschenrechner - Tagebuch - Notizen - Tabellenkalkulation - Adressenverzeichnis - Banking
Reisen	**Dienstprogramme**
- Stadtführer - Währungsrechner - Übersetzer - GPS, Karten - Routen, Fahrpläne - Wettervorhersage	- Profilmanager - Bildschirmschoner - Adressbuch - Prozessmanager - Anrufmanager - Dateimanager

Tab. 4.1: Kategorien mobiler Applikationen (nach MOBILE MARKETING ASSOCIATION, 2010)

In die Kategorie der Reisen fallen die für diese Arbeit relevanten touristischen Applikationen, welche für die mobile Kartographie ein Einsatzgebiet mit stetig wachsendem Potential sind. Diese touristischen Applikationen stellen im mobilen Technologie- und Informationszeitalter eine zeitgemäße Form der Reiseinformation und des Reiseservices dar und können der Tourismusbranche als neuartiges Marketinginstrument im sich verschärfenden Wettbewerb dienen.

Aus der Mobilität von Reisenden sowie aus dem Bedürfnis, unterwegs auf einer Reise in einer fremden Umgebung Informationen einholen zu können, resultiert das erwartete hohe Marktpotential mobiler touristischer Dienste. Dieser Informationsbedarf ist besonders groß unter Spontan- und Individualreisenden, welche den Trend aktueller Tourismusentwicklungen bestimmen. Tendenziell werden heutzutage mehr Medien in die Reisevorbereitung und -planung sowie in die Reise selbst eingebunden. Das Angebot hierzu ist bereits weit entwickelt. Abgesehen von Reservierungs- und Buchungsdiensten bietet der Bereich der Information eine große Bandbreite an interaktiven Stadt- oder Wanderführern, einfachen touristischen SMS-Informationen bis hin zu digitalen Karten, Navigationsdiensten und LBS. Es ist vor allem der hohen Aktivität von Mobilfunkanbietern, großen Systemherstellern und Reisemedienunternehmen zuzuschreiben, dass dieses weite Angebot existiert. Destinationen als Anbieter spielen auf dem Markt bisher nur eine untergeordnete Rolle (JÄGER, 2005).

4.1 Analyse bestehender touristischer Applikationen

Für die vorliegende Arbeit wurden fünf Applikationen für iPhone untersucht (*Lonely Planet Berlin*, *Lonely Planet Paris*, *Ostseeküste - ADAC Wanderführer*, *Allgäu*, *The North Face Trailhead*) sowie vier Applikationen für Smartphones mit einem Android-Betriebssystem (*outdooractive*, *TravelBook Berlin*, *Vienna Travel Guide*, *Qype*), die in Tabelle 4.2 kurz vorgestellt werden sollen. Ihnen allen ist gemein, dass sie aus dem touristischen Bereich stammen und dass Karten ein Teil der Anwendung sind.

Android	
outdooractive	Der App der Internetplattform *outdooractive.com* ermöglicht es dem Benutzer, Wander-, Rad-, Ski-, Schneeschuh-, Rodel- u. viele Touren anderer Sportarten über eine Umkreis- oder Textsuche von *outdooractive.com* herunterzuladen und anzusehen.
Qype	Der App der gleichnamigen Online-Community bietet über eine Million nutzergenerierte Bewertungen und Empfehlungen. Über 850 Kategorien in 159.000 Städten weltweit machen es möglich, die besten Restaurants, Bars, Clubs, Geschäfte, Dienstleistungen u.v.m. in unmittelbarer Nähe zu finden.
TravelBook Berlin	Eine Installation des Apps *TravelDroyd* ist erforderlich, um Stadtführer-Applikationen verschiedener Städte, wie z.B. *TravelBook Berlin,* nutzen zu können. Letztere liefert umfangreiche touristische Informationen und Kartenmaterial.
Vienna Travel Guide	Dieser Stadtführer bietet zahlreiche Hintergrundinformationen, Kartenmaterial und viel Wissenswertes für Besucher Wiens.
iPhone	
Allgäu	Neben einer Wanderkarte im Maßstab 1:25.000 für ganz Deutschland und Österreich bietet dieser App eine Auswahl an Touren der unterschiedlichsten Sportarten mit detaillierten Beschreibungen, Wegverläufen, Bildern und POIs sowie Nachrichten zur Region, den lokalen Wetterbericht und aktuelle Veranstaltungen.
Lonely Planet Berlin	Dieser Stadtführer für Berlin ist an die Gliederung der *Lonely Planet*-Reiseführer in Buchform angelehnt. Er hält eine interaktive Karte von Berlin und Umgebung bereit, empfiehlt viele Sehenswürdigkeiten und Aktivitäten und enthält Informationen und Bilder.
Lonely Planet Paris	Der *Lonely Planet*-App für Paris ist anders aufgebaut als der für Berlin, hat aber die gleichen Eigenschaften: er hält eine interaktive Karte von Paris bereit, empfiehlt viele Sehenswürdigkeiten und Aktivitäten und enthält zahlreiche Informationen und Bilder.
Ostseeküste - ADAC Wanderführer	Der Wanderführer von ADAC für die Ostseeküste liefert Tourenbeschreibungen, Höhenprofile, Tourenverläufe in Wanderkarten mit Höhenlinien, dem gesamten Wegenetz und topographischen Inhalten, Bildern und POIs entlang der Tour und bietet eine *Augmented Reality.*
The North Face Trailhead	Mit *The North Face Trailhead* kann der Benutzer *Trails* für verschiedene Sportarten auf der ganzen Welt finden und kann seine eigenen Routen aufzeichnen und veröffentlichen. Der App nutzt die *EveryTrail-Technologie*, um dynamisch und standortabhängig Stre-

	cken wie Wanderwege, Bike-Routen, Skipisten und vieles mehr anzuzeigen.

Tab. 4.2: Untersuchte touristische Applikationen (ANDROIDPIT, ITUNES)

4.1.1 Analyse des Funktionsumfangs

Bei der Analyse des Funktionsumfangs der oben aufgeführten Anwendungen kristallisierten sich zehn Fragestellungen heraus, anhand derer die Applikationen untereinander verglichen werden sollen.

1) Wird die Position des Benutzers berücksichtigt?
2) Sind die Karte und andere Inhalte miteinander verknüpft?
3) Wird eine vom Entwickler selbst erstellte Karte verwendet?
4) Besteht die Möglichkeit, das Kartendesign zu ändern?
5) Besteht die Möglichkeit, Karteninhalte ein- und auszublenden?
6) Sind neben Zoomen und Verschieben weitere Interaktionen mit der Basiskarte möglich?
7) Können die Inhalte durch den Benutzer personalisiert werden?
8) Besteht die Möglichkeit der Dateneingabe?
9) Verfügt die Applikation über eine Suchfunktion?
10) Verfügt die Applikation über eine interne Routing-Funktion?
11) Wird mit einer *Augmented Reality* gearbeitet?
12) Kann die Applikation auch ohne Internetverbindung uneingeschränkt verwendet werden?

Tabelle 4.3 gibt einen kurzen Überblick über die Funktionen der untersuchten Applikationen, auf die im Folgenden näher eingegangen werden soll. Die Zahlen in der Kopfzeile entsprechen der Nummerierung der oben aufgeführten Fragestellungen.

	1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**	**11.**	**12.**
Android												
outdooractive	✓	✓	–	✓	–	–	–	–	✓	–	–	–
Qype	✓	✓	–	–	–	–	✓	✓	✓	–	–	–
TravelBook Berlin	✓	✓	✓	✓	✓	✓	✓	–	✓	✓	–	–
Vienna Travel Guide	–	✓	✓	–	✓	–	–	–	✓	–	–	✓

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.
iPhone												
Allgäu	✓	✓	✓	✓	–	–	–	–	–	–	–	–
Lonely Planet Berlin	✓	✓	✓	–	✓	–	–	–	–	–	–	✓
Lonely Planet Paris	✓	✓	✓	–	✓	–	✓	–	✓	–	–	✓
Ostseeküste - ADAC Wanderführer	✓	✓	✓	–	–	–	–	–	–	–	✓	–
The North Face Trailhead	✓	✓	–	✓	–	–	✓	✓	✓	✓	–	–

Tab. 4.3: Funktionsumfang der untersuchten Applikationen

Außer *Vienna Travel Guide* berücksichtigen alle Applikationen den aktuellen Standort des Benutzers: er wird in der enthaltenen Karte angezeigt und teilweise kann die Entfernung des Benutzers zu verschiedenen Objekten für Sortierungen oder Suchanfragen verwendet werden.

Eine Verknüpfung von Objekten in der Karte zu anderen Inhalten der Anwendung existiert in jeder der oben genannten Applikationen, jedoch liegen diese Verknüpfungen in verschiedener Gestalt vor. In jedem Fall ist es möglich, Routen oder POIs aus einer Liste auszuwählen und anschließend in einer Karte angezeigt zu bekommen. Ebenso können Objekte in der Karte ausgewählt werden, woraufhin ein Aufruf von Informationen dazu erfolgt, allerdings in verschiedenem Umfang. Teilweise erscheint nur der Name des Objektes oder ein auch Foto, teilweise handelt es sich um eine Verknüpfung zu einer anderen Stelle im App, die sehr detaillierte Informationen bereithält. Nur in zwei Fällen (*Lonely Planet Berlin* und *Lonely Planet Paris*), befinden sich auch in Fließtexten Links zu den entsprechenden Objekten in der Karte.

Sechs der untersuchten neun Applikationen verwenden eigenes Kartenmaterial, alle anderen greifen ausschließlich auf *Google Maps* zurück. Damit geht einher, dass das bereitgestellte eigene Kartenmaterial auch ohne Internetverbindung verwendet werden kann, was einen klaren Vorteil darstellt. *Allgäu* verwendet neben den eigenen Karten auch wahlweise Karten von *Google Maps* und ist somit nur bedingt auf eine Internetverbindung angewiesen bzw. ist eine uneingeschränkte Nutzung ohne Internetzugang nicht möglich ist.

Dieser Gesichtspunkt leitet bereits zu der Fragestellung über, ob dem Benutzer verschiedene Kartendesigns zur Auswahl stehen. Bei der zuletzt genannten Applikation ist dies auf Grund der verschiedenen Datenquellen der Fall sowie generell bei Applikationen, die *Google Maps* verwenden, da dies standardmäßig verschiedene Ansichten zur Verfügung stellt (Straßenkarte, Satellitenbild, Hybrid, Terrain), wobei nicht jede Ansicht in jedem App Verwendung findet. *TravelBook Berlin* bietet sogar drei eigene Kartendesigns: eine Straßen-, eine Wander- und eine Nachtkarte in dunklen Farben, um ein Blenden des Benutzers durch ein zu helles Display bei Nacht zu vermeiden.

Eine weitere Interaktion mit der Karte, die nur vier der neun untersuchten Applikationen zur Verfügung stellen, ist das Ein- und Ausblenden von Karteninhalten. In *Vienna Travel Guide*, *Lonely Planet Berlin* und *Lonely Planet Paris* können Signaturen von POIs kategorienweise ein- und ausgeblendet werden, wohingegen *TravelBook Berlin* zusätzlich das wahlweise Anzeigen von Höhenlinien und georeferenzierten *Wikipedia*-Artikeln zu Objekten in der Karte erlaubt.

Neben den ‚Standard'-Interaktionen Zoomen und Verschieben sind nur in einem der untersuchten Apps weitere Interaktionen mit der Basiskarte möglich: in *TravelBook Berlin* kann diese zusätzlich gedreht und auch wieder genordet werden.

Vier von neun Apps ermöglichen es dem Benutzer, Inhalte zu personalisieren. Mit *Qype*, *Lonely Planet Paris* und *The North Face Trailhead* können einzelne Objekte (z. B. Restaurants) bzw. Routen als Favoriten markiert werden, um so ein schnelleres Auffinden zu gewährleisten. *TravelBook Berlin* erlaubt es, in der Karte eigene Pins und Marker zu setzen und sie mit Informationen zu versehen.

Die Möglichkeit der Dateneingabe bieten zwei der untersuchten Applikationen, jedoch ist hier in jedem Fall ein Login erforderlich. Dies hängt damit zusammen, dass jene Apps ihre Inhalte aus Online-Communities beziehen. *Qype* greift auf die gleichnamige Online-Plattform zu und *The North Face Trailhead* nutzt Touren von *EveryTrail*. Jede dieser Plattformen basiert auf nutzergenerierten Inhalten, somit kann auch der Benutzer des Apps die Möglichkeit wahrnehmen, solche auf der jeweiligen Plattform zu erstellen. Dies kann auch direkt innerhalb der Apps geschehen. Mit *outdooractive* ist das Erzeugen und Veröffentlichen nutzergenerierter Inhalte prinzipiell ebenfalls möglich, da auch dieser App *User Generated Content* der gleichnamigen Online-Community verwendet, jedoch muss dafür die Internetseite der Plattform genutzt werden; im App selbst besteht diese Möglichkeit nicht.

Sechs der Applikationen bieten dem Benutzer eine Suchfunktion an. In *Lonely Planet Paris* und *Vienna Travel Guide* ist es möglich, die gesamten Inhalte des Apps nach einem Begriff zu durchsuchen. In *Qype* kann anhand von Namen oder Kategorien nach POIs gesucht werden, in *TravelBook Berlin* auch nach Straßennamen. *outdooractive* und *The North Face Trailhead* enthalten eine Suchfunktion, um spezielle Routenvorschläge zu finden. In ersterem App kann die Suche nach Sportart, Umkreis oder als freie Suche erfolgen. In *The North Face Trailhead* ist die Suchfunktion sehr umfangreich: es kann nach Stadt oder Postleitzahl, Suchradius, Sportart und/oder freien Suchbegriffen und wahlweise nur nach Routen, die mit Fotos versehen sind, gesucht werden.

Über eine interne Routing-Funktion verfügen bloß *TravelBook Berlin* und *The North Face Trailhead*, wobei letztere Applikation auf Berechnungen von *Google Maps* zurückgreift, die Ergebnisse allerdings im App angezeigt. *ADAC Wanderführer* und *Allgäu* hingegen bieten nur einen Link zu der Routenberechnung von *Google Maps*.

Lediglich die Anwendung *ADAC Wanderführer* bietet als einziger App eine *Augmented Reality*, mit der umliegende Berggipfel an der Ostseeküste bestimmt werden können.

Ohne eine Internetverbindung können nur drei der neun untersuchten touristischen Applikationen uneingeschränkt genutzt werden. Vor allem *Qype*, *outdooractive* und *The North Face Trailhead* können ohnehin nicht offline Verwendung finden, da hier sowohl auf Karten als auch auf Tourenvorschläge oder POIs ohne Internetzugang nicht zugegriffen werden kann. Eine nur leichte

Einschränkung hingegen durch eine fehlende Internetverbindung erfahren *Allgäu* und *ADAC Wanderführer*: *Allgäu* kann die als Darstellungsalternative angebotenen Karten von *Google Maps* nicht anzeigen und *ADAC Wanderführer* seine Routing-Funktion nicht nutzen. Ebenso *TravelBook Berlin* braucht für eine Routenberechnung einen Internetzugang. Von den *Wikipedia*-Artikeln in der Karte dieses Apps liegen einige nicht offline vor, stattdessen wird die online-Variante angeboten.

Da sich die oben formulierten Kriterien, hinsichtlich derer die neun Applikationen untersucht wurden, jeweils mit ‚Ja' oder ‚Nein' beantworten lassen, kann berechnet werden, wie viel Prozent der Kriterien ein App erfüllt. An dieser Stelle muss jedoch darauf hingewiesen werden, dass diese Kriterien keinesfalls allgemeingültig sind, da nur eine vergleichsweise winzige Stichprobe aus der stetig wachsenden Gesamtmenge aller touristischen Applikationen entnommen wurde, auch wenn sich viele Applikationen in Aufbau und Funktionalitäten ähneln.

Tabelle 4.4 zeigt die jeweilige Anzahl der durch die untersuchten Applikationen erfüllten Kriterien und die sich daraus ergebende Prozentzahl.

	erfüllte Kriterien	**nicht erfüllte Kriterien**	**erfüllte Kriterien in Prozent**
Android			
outdooractive	4	8	33%
Qype	5	7	42%
TravelBook Berlin	9	3	75%
Vienna Travel Guide	5	7	42%
iPhone			
Allgäu	4	8	33%
Lonely Planet Berlin	5	7	42%
Lonely Planet Paris	7	5	58%
Ostseeküste - ADAC Wanderführer	4	8	33%
The North Face Trailhead	7	5	58%

Tab. 4.4: Durch die untersuchten touristischen Applikationen erfüllte Kriterien

Nur drei der neun Applikationen erfüllen mehr als 50% der Kriterien: *The North Face Trailhead* und *Lonely Planet Paris* erreichen jeweils 58%, *TravelBook Berlin* sogar 75%. Je höher diese Prozentzahl ist, umso mehr Interaktionen und Funktionalitäten stehen dem Benutzer zur Verfügung. Jedoch sagt diese Zahl beispielsweise nichts über Qualität oder Quantität der Inhalte, über die Zweckmäßigkeit der Menüführung, über das Vorhandensein einer Hilfsfunktion und vieles mehr aus. Ebenso muss beachtet werden, dass nicht jedes erfüllte Kriterium gleichgewichtig bzw. immer positiv ist. So kann denn davon ausgegangen werden, dass Besitzer eines Smartphones in ihrem Mobilfunkvertrag über eine Internetflatrate verfügen und somit nicht unbedingt auf einen App, der auch ohne Internet uneingeschränkt funktioniert, angewiesen sind; zumal Apps, die auf Internetinhalte zugreifen, in der

Regel einen umfangreicheren Inhalt bieten. Befindet sich der Benutzer jedoch im Ausland, können hohe Verbindungskosten entstehen, wenn doch ein Internetzugang vonnöten ist. Des Weiteren könnte eben jenes Kriterium der Unabhängigkeit vom Internet überdacht werden. Während der Analyse wurde auf eine uneingeschränkte Nutzung der Applikation ohne Internet geachtet, jedoch stellt sich die Frage, ob eine externe Routing-Funktion (wie etwa in Form eines Links zu *Google Maps* bei *ADAC Wanderführer* oder *Allgäu*) zum kompletten Funktionsumfang zählt wie bei der Analyse angenommen, wird sie doch im App als eigenständiger Menüpunkt aufgeführt. Ebenso wenig wird mit dieser Prozentzahl beachtet, welchen Zweck eine Applikation erfüllen soll und ob für diesen Zweck all jene Kriterien von Bedeutung sind. Somit muss ein niedriger Wert nicht gleichbedeutend sein mit wenig Zweckmäßigkeit. Die hier untersuchten Applikationen versuchen jedoch, ein breites Publikum abzudecken und sind wenig spezifisch.

Überraschenderweise wird die höchste Prozentzahl von einer Applikation erreicht, die vom Design her auf den ersten Blick weniger professionell und modern wirkt. Offenbar lassen sich Benutzer sehr von einem professionellen und modernen Design beeindrucken und assoziieren dies mit einem hohen Funktionalitätsumfang.

Eine tabellarische Zusammenfassung der oben beschriebenen Funktionalitäten aller untersuchten Applikationen befindet sich in Anhang XI und XII.

4.1.2 Allgemeine Klassifizierung der untersuchten Applikationen

Die eben erwähnte geringe Aussagekraft der oben errechneten Prozentwerte macht es schwierig, eine Klassifizierung von touristischen Applikationen hinsichtlich ihres Funktionsumfangs vorzunehmen. Vielmehr bietet sich hierfür eine allgemeinere Charakteristik der Applikationen bezüglich ihrer Zweckbestimmung und ihrem räumlichen Geltungsbereich an. Die für die vorliegende Arbeit untersuchten Applikationen lassen sich klar in Stadt- und in Freizeitführer trennen. Freizeitführer bieten Tourenvorschläge an, um ein Gebiet in Form einer sportlichen Aktivität wie Wandern, Radfahren und ähnliches. zu erkunden. Hierbei ist nicht auszuschließen, dass Freizeitführer auch Sightseeing-Touren für Städte vorschlagen. Stadtführer hingegen stellen eine Stadt vor, indem sie für Touristen interessante Anlaufpunkte wie Sehenswürdigkeiten, Museen, Restaurants etc. nennen, in der Regel jedoch ohne diese in Form einer Route zu verbinden. Der Geltungsbereich der Applikationen kann in regional und global unterschieden werden, d.h. ob die mobile Anwendung nur für einen spezifischen Ort oder eine Region oder wesentlich weitreichender ausgelegt ist. Daraus ergeben sich vier Klassen für mobile, touristische Applikationen: regionale Stadtführer, regionale Freizeitführer, globale Stadtführer und globale Freizeitführer. Eine Zuordnung der untersuchten Applikationen befindet sich in Tabelle 4.5.

	Räumlicher Geltungsbereich	Zweckbestimmung
Android		
outdooractive	global	Freizeitführer
Qype	global	Stadtführer
TravelBook Berlin	regional	Stadtführer
Vienna Travel Guide	regional	Stadtführer
iPhone		
Allgäu	regional	Freizeitführer
Lonely Planet Berlin	regional	Stadtführer
Lonely Planet Paris	regional	Stadtführer
Ostseeküste - ADAC Wanderführer	regional	Freizeitführer
The North Face Trailhead	global	Freizeitführer

Tab. 4.5: Klassifizierung der untersuchten touristischen Applikationen

Jede der vier Klassen ist hier vertreten: vier regionale Stadtführer, zwei regionale Freizeitführer, ein globaler Stadtführer und zwei globale Freizeitführer. Die Applikationen mit globalem Geltungsbereich sind ausschließlich jene, die ihre Informationen komplett aus dem Internet beziehen; eine Internet-unabhängige Lösung würde zu viel Speicherkapazität in Anspruch nehmen. Sicher sind die globalen Applikationen nicht wirklich global, d.h. sie bieten nicht für jeden Ort weltweit Informationen, aber sie gehen definitiv über einen regionalen Geltungsbereich hinaus. Des Weiteren basieren sie alle auf nutzergenerierten Inhalten, somit besteht zumindest theoretisch die Möglichkeit einer weltweiten Abdeckung mit Informationen. Nutzergenerierte Inhalte sind der realistischste Weg eben diese Globalität zu realisieren; eine Datenakquisition durch den Entwickler oder Herausgeber selbst würde ein solches Projekt nahezu unmöglich machen. Die regionalen Applikationen verraten ihre Eigenschaft der Regionalität bereits durch ihren Namen. Es handelt sich dabei um jene, deren Inhalte offline vorliegen (abgesehen von aktuellen Wetter- und Veranstaltungsinformationen der Applikation *Allgäu*). Im Gegensatz zu den globalen Applikationen sind die Inhalte durch den begrenzten räumlichen Geltungsbereich weniger umfangreich und müssen deswegen nicht zwingend aus dem Internet bezogen werden.

4.2 Theoretische Grundlagen für die Konzeption und Entwicklung einer mobilen Applikation

Im praktischen Teil dieser Arbeit soll eine eigene mobile Kartenapplikation für touristische Aktivitäten in der Pilotregion Sächsische Schweiz konzipiert und prototypisch implementiert werden. Das laut Aufgabenstellung hierfür vorgesehene mobile Endgerät ist das Smartphone *G2 Touch*. Dieses

Gerät hat eine Bildschirmgröße von 320 x 480 Pixeln, verfügt über eine interne GPS-Antenne und das Betriebssystem Android.

4.2.1 Die Software-Plattform Android

Android stellt sowohl ein Betriebssystem als auch eine Software-Plattform für mobile Endgeräte wie Mobiltelefone, Smartphones und Netbooks dar. Android ist eine quelloffene und freie Software und wurde von der Open Handset Alliance auf Basis des *Linux-Kernel 2.6* entwickelt. Dieser ist für die Speicher- und Prozessverwaltung sowie für die Netzwerkkommunikation zuständig und stellt die Gerätetreiber für das System bereit. Weitere wichtige Bestandteile sind die auf Java-Technik basierende virtuelle Maschine *Dalvik* und die dazugehörigen Android-Java-Klassenbibliotheken (MOSEMANN/KOSE, 2009). Für das Programmieren von eigenen Android-Applikationen stehen insgesamt 1448 Java-Klassen und 394 Schnittstellen zur Verfügung. 511 Klassen und 128 Schnittstellen davon sind Android-spezifisch. Aktuell ist die Android-Plattform 3.0, was dem API Level 11 entspricht, verfügbar.

Android wird über einen Touchscreen bedient und definiert eine Reihe von Hardware-Tasten. Die Mindestanforderung bilden die Tasten *Home*, *Menü* und *Zurück* (WIKIPEDIA).

Bei Android werden verschiedene Begriffe der Oberflächengestaltung unterschieden. Eine spezielle Java-Klasse, deren Aufgabe die Darstellung von Texten, Schaltflächen und Menüoptionen ist und die auf Eingaben des Benutzers reagiert, wird *Activity* genannt. Pro Bildschirmseite muss eine *Activity* implementiert werden. *Activities* werden von der Klasse `android.app.Activity` abgeleitet. Neben den standardmäßigen Java-*Activities* werden durch die Android-API *Activities* für verschiedene Aufgaben bereitgestellt. Die darstellbaren Elemente einer Bildschirmseite werden als *View* bezeichnet und sind von der Klasse `android.view.View` abgeleitet. Dabei gibt es *View*s, die einen Rahmen für andere Views darstellen. Eine *View* zum Anzeigen einer Tabelle enthält zum Beispiel Views, die als Textfelder dienen. Solche Rahmen-Views wie Tabellen werden *Viewgroup* genannt und entstammen der Klasse `android.view.Viewgroup`. Die Elemente dieser Klasse werden unterschieden in *Viewgroups*, die selbst als Formularelemente dienen und in *Viewgroup*s, die im Hintergrund die Anordnung, also das Layout, ihrer untergeordneten *Views* festlegen. Demzufolge ist eine Bildschirmseite eine Baumstruktur von *Viewgroup*s und *Views*. Die Wurzel stellt in der Regel eine Layout-*Viewgroup* dar, die den Rahmen bildet für die restlichen Oberflächenelemente. Die Android-API hält ein breitgefächertes Angebot an *Views* für weitestgehend alle Anwendungsfälle bereit. Bei Bedarf können auch eigene *Views* programmiert werden. Um die vorhandenen Möglichkeiten für die spätere Konzeption einer eigenen Anwendung zu demonstrieren, zeigt Anhang XIII zeigt die Basislayout-*Views* der Android-API.

Die Verbindung einer *Activity* mit einer *View* definiert einen Bildschirmdialog. Über diesen tritt der Benutzer mit der Applikation in Kontakt und erhält die Kontrolle über sie.

Für jede *View* existiert eine Java-Klasse, aber auch ein entsprechendes XML-Element. Somit kann das Layout einer Bildschirmseite sowohl in einer XML-Datei als auch im Java-Code definiert werden. Eine Kombination von beidem ist ebenso möglich. Aus Gründen der Übersichtlichkeit wird die Verwen-

dung von XML empfohlen. Die Java-Komponenten hingegen werden für Zugriffe auf Elemente der Bildschirmseite aus dem Programmcode heraus genutzt, wie zum Beispiel für das Auslesen von Eingabefeldern.

Aus diesem Nebeneinander von Java und XML ergibt sich eine strenge Ordnerstruktur beim Anlegen einer eigenen Android-Applikation. Die Ordnerstruktur einer Android-Applikation mit Namen ‚Wanderfuehrer' ist in Abbildung 4.2 zu sehen.

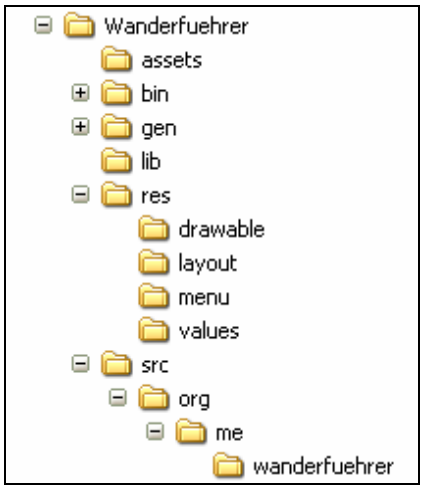


Abb. 4.2: Beispiel für die Ordnerstruktur eines Android-Projektes

Die Ordner `bin`, und `gen` sind an dieser Stelle nicht von Interesse, da sie von der Entwicklungsumgebung erstellt wurden und solche automatisch erstellten Ordner je nach Entwicklungsumgebung variieren.

Im Ordner `src` liegt der Java-Quellcode der Applikation. Aus der Benennung der untergeordneten Verzeichnisse ergibt sich der *Package*-Name der Anwendung, also in diesem Falle `org.me.wanderfuehrer`. Dieser Name dient als eindeutiger Bezeichner einer Applikation.

Der Ordner `res` enthält Ressourcen, die je nach Ressourcenart in einem bestimmten Unterverzeichnis abgelegt werden. Im Unterordner `drawable` befinden sich Graphikdateien; `layout` enthält XML-Dateien, die das Layout einer Bildschirmseite festlegen; in den Ordner `menu` gehören XML-Dateien, die Menüs definieren und ebenso im Ordner `values` werden XML-Ressourcen abgelegt. In letzterem wird wieder je nach Ressourcenart unterschieden, wie die Ressourcen angesprochen werden: `string` betrifft Zeichenketten, ebenso wie `array`, allerdings hier in Form von *Arrays*; als `color` werden Farbressourcen definiert und `style` umfasst *Styles* und *Themes*. Ein *Style* ist eine Gruppe von Attributwerten (ähnlich wie in CSS), die das Aussehen einzelner *Views* definiert; *Themes* hingegen legen das Aussehen kompletter Bildschirmseiten fest. Es gibt weitere Ordner, die im `res`-Verzeichnis existieren können, innerhalb der zu entwickelnden Applikation jedoch nicht zur Anwendung kommen (BECKER/PANT, 2009).

Im Verzeichnis `assets` können Dateien jeglicher Art abgelegt werden. Der Unterschied zu den anderen Ressourcen ist hierbei, dass dieser Ordner wie ein Dateisystem funktioniert, d.h. die Inhalte können gelistet, iteriert, in Unterordnern abgelegt werden etc. und es wird über ihre originalen Dateinamen auf sie zugegriffen. Jar-Dateien, die als externe Bibliotheken in die Applikation eingebunden werden, werden im `lib`-Verzeichnis gespeichert (ANDROID DEVELOPERS).

Im Wurzelverzeichnis dieser Ordnerstruktur befindet sich stets die Datei `AndroidManifest.xml`, in welcher der *Package*-Name der Applikation festgelegt wird, alle Komponenten der Applikation wie *Activities* und *Services* aufgelistet und Berechtigungen auf Anwendungs- und Prozessebene vergeben werden. Erst durch diese Datei wird aus der Ordnerstruktur und ihren Inhalten eine lauffähige Anwendung.

Eine weitere Datei von großer Wichtigkeit liegt (im Falle von Eclipse als Entwicklungsumgebung) im Order gen, nämlich die automatisch generierte die Datei `R.java`. Diese enthält Schlüssel für alle in der Anwendung definierten Ressourcen. Anhand dieser Schlüssel, die aus Ressourcenart und Ressourcennamen abgeleitet werden, kann auf Ressourcenobjekte zugegriffen werden. Im Java-Quellcode wird beispielsweise eine Farbe, die als ‚rot' definiert ist, über den Bezeichner `R.color.rot` angesprochen werden; innerhalb einer XML-Datei hingegen über `@color/rot`.

Das Ergebnis eines Android-Projektes ist die eigentliche Applikation in Form einer `.apk`-Datei. Diese wird vom Compiler der jeweiligen Entwicklungsumgebung erzeugt und enthält nur wirklich benötigte Klassen (BECKER/PANT, 2009).

Als besondere `drawable`-Ressourcen, die als Hintergrund für einige *Views* dienen, verwendet Android sogenannte *NinePatch*-Dateien. Dabei handelt es sich um eine PNG-Bilddatei mit der Endung `.9.png`, in welcher Bereiche definiert sind, die automatisch skaliert werden, sobald der Inhalt einer *View* die normalen Bildgrenzen überschreitet. Abbildung 4.3 zeigt eine solche *NinePatch*-Datei für einen Button. Die linke und die obere schwarze Linie bestimmen den Bereich, der skaliert wird, Bereiche außerhalb verändern ihre Größe nicht; die untere und die rechte Linie legen die Platzierung von Text innerhalb des Bildes fest. Diese Linien sind bei der Visualisierung der *NinePatch*-Dateien innerhalb der Applikation nicht sichtbar, es sei denn, das Bild wird verkleinert. Aus diesem Grund sowie anderen Darstellungsfehlern, die dabei auftreten können, empfiehlt sich eine Unterschreitung der eigentlichen Bildgröße nicht. Wenn bei der Entwicklung einer Android-Applikation nicht das Standard-Layout verwendet werden soll, kann es erforderlich sein, eigene *NinePatch*-Dateien zu erstellen, beispielsweise für Buttons, Expander etc. (ANDROID DEVELOPERS).

Abb. 4.3: NinePatch-Datei eines Buttons

4.2.2 Lebenszyklus einer Activity

Android unterscheidet sich zu der Mehrzahl anderer Handy-Betriebssysteme darin, dass es den knapp dimensionierten Speichern und Prozessoren von Smartphones Rechnung trägt, indem es in die Lebensdauer von Prozessen eingreift und diese bei knappen Ressourcen beendet, was sich auf die darin laufenden Anwendungen und deren Komponenten (*Activities*, *Services*, *Broadcast Receivers*) auswirkt. Jede Komponente verfügt über verschiedene Zustände, welche man zusammenfassend als Lebenszyklus bezeichnet. Diese Lebenszyklen unterliegen der Kontrolle des Betriebssystems.

An dieser Stelle soll der Lebenszyklus einer *Activity* beleuchtet werden. Von ihrer Erzeugung bis zur Zerstörung durchläuft eine *Activity* verschiedene Stadien, für welche jeweils eine Methode in der Klasse `android.app.Activity` existiert. Diese Methoden können in der eigenen Implementierung überschrieben werden. Der gesamte Lebenszyklus ist in Abbildung 4.4 zu sehen.

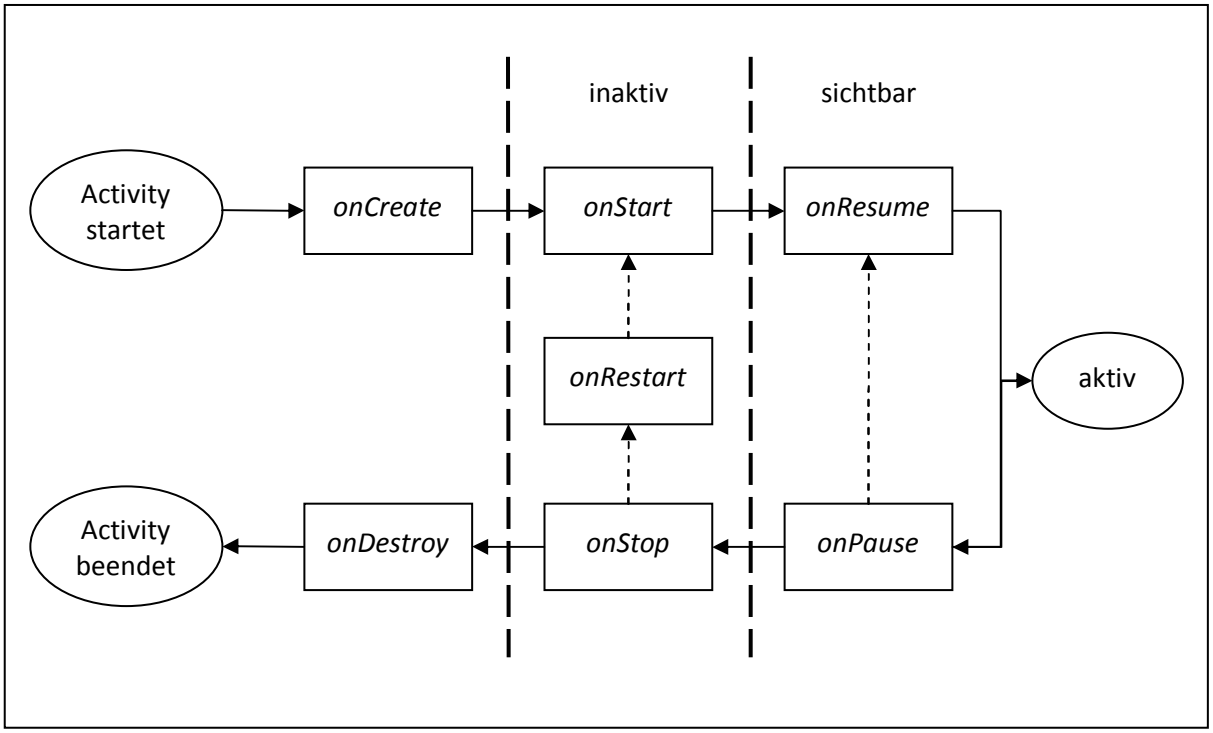


Abb. 4.4: Lebenszyklus einer Activity (nach BECKER/PANT, 2009)

Durch die `onCreate`-Methode wird eine *Activity* ins Leben gerufen. Diese Methode wird genutzt, um Layouts, Schaltflächen und Menüs zu initialisieren. Bevor die *Activity* jedoch auf dem Bildschirm angezeigt wird und als ‚aktiv' gilt, d.h. auf Interaktionen des Benutzers reagiert, durchläuft sie außerdem die Methoden `onStart` und `onResume`. Wird die *Activity* nun von einer weiteren *Activity* überlagert, (d.h. sie ist zwar noch sichtbar, reagiert aber nicht mehr auf Eingaben des Benutzers) kommt die Methode `onPause` zum Einsatz. Sobald diese überlagernde *Activity* beendet wird, wird wieder `onResume` aufgerufen. Ist die Überlagerung durch eine andere *Activity* jedoch komplett, so dass die *Activity* inaktiv und nicht mehr sichtbar ist, erfolgt ein Aufruf von `onStop`. Beim Beenden dieser komplett überlagernden *Activity* werden hingegen `onRestart`, `onStart` und `onResume` auf-

gerufen. Ein Aufruf von `onDestroy` erfolgt nur, wenn die *Activity* aus dem Cache entfernt und ihre Ressourcen somit freigegeben werden (BECKER/PANT, 2009).

4.2.3 Design-Guidelines für Android-Applikationen

Auf der offiziellen Website des Android-SDK werden Richtlinien beschrieben, die bei der Entwicklung einer Android-Anwendung Berücksichtigung finden sollten.

Neben Guidelines für den Aufruf von *Activities* existieren Richtlinien, die die Gestaltung von Icons und Menüs einer Android-Applikation betreffen. Bei Android wird zwischen zwei Arten von Menüs unterschieden: zwischen Options- und Kontextmenüs. Optionsmenüs sind Menüs, die über die *Menü*-Taste des Smartphones aufgerufen werden. Kontextmenüs werden durch ein langes Gedrückthalten von Bildschirmobjekten aktiviert. Für jede *View* kann ein Kontextmenü deklariert werden, pro *Activity* gibt es hingegen nur ein Optionsmenü. Ein Optionsmenü enthält Befehle, die global für die aktuelle *Activity* angewendet werden können oder die eine andere *Activity* aufrufen. Sie beziehen sich nicht auf ein einzelnes ausgewähltes Element. Letzteres wird durch das Kontextmenü bedient, das sich mit einem Rechts-Klick am PC vergleichen lässt. Es ähnelt einem Direktaufruf, der Befehle dupliziert, die auch anderweitig aufgerufen werden können. Ein Kontextmenü ist also eine Auflistung von Aktionen, die mit dem gewählten Element ausgeführt werden können. Da Kontextmenüs oft nur schwer von Benutzern entdeckt werden, weil es keine explizite Schaltfläche oder ähnliches für deren Aufruf gibt, sollten sie wirklich nur Befehle enthalten, die auch anderweitig ausgeführt werden können. Ausnahmen bilden Befehle wie ‚Text auswählen', die beispielsweise bei Textfeldern selbstständig vom System hinzugefügt werden.

Natürlich müssen Befehle nicht nur in Menüs zur Auswahl stehen, sie können auch als Schaltfläche direkt auf dem Bildschirm sichtbar sein; somit sind sie leichter und schneller auffindbar. Dies muss aber abgewogen werden gegen den Platz, den solche Benutzerkontrollen auf dem ohnehin schon kleinen Bildschirm einnehmen. Anhand dieser Unterschiede zwischen Options- und Kontextmenüs, können Befehle einem Menü entsprechend zugeordnet werden. Auf Grund der begrenzten Bildschirmgröße können unter Umständen nicht alle Menüeinträge komplett angezeigt werden. Deswegen empfiehlt es sich, die wichtigsten Befehle an erster Stelle im Menü aufzuführen und ähnliche Befehle nah beieinander anzuordnen. Speziell für das Kontextmenü bedeutet das: das Berühren eines Elements des Bildschirminhalts sollte denselben Befehl aktivieren wie das Berühren dessen ersten Kontextmenüeintrags. Wenn der Benutzer eine Android-Anwendung ohne den Aufruf des Kontextmenüs komplett bedienen kann, dann ist sie optimal gestaltet. Des Weiteren sollte ein Kontextmenü immer den Namen des Elementes enthalten, auf das es sich bezieht. Durch das Verlegen von Befehlen in ein Menü wird Platz auf dem Bildschirm frei, andererseits sind auf der aktuellen Bildschirmseite fixierte Befehle auffälliger und somit einfacher zu benutzen. Deshalb sollten dies nur die wichtigsten Befehle sein.

Es gibt weitere Guidelines bezüglich *Widgets*. Ein *Widget* zeigt die wichtigsten Informationen einer Anwendung auf dem *Home*-Bildschirm (ähnlich dem Desktop eines PCs) des Smartphones an. Ein Standard-Android-*Widget* ist zum Beispiel die Analoguhr. Da für die im Rahmen dieser Arbeit zu kon-

zipierende Applikation kein *Widget* geplant ist, wird von einer Erläuterung der entsprechenden Richtlinien abgesehen. Von Interesse sind hingegen die Guidelines für Android-Icons. Seit Android 1.6 werden verschiedene Bildschirmauflösungen von Smartphones abgedeckt, indem alle Icons einer Anwendung in drei Auflösungen vorliegen: *hdpi* (high, > 180 dpi), *mdpi* (medium, 120 - 180 dpi) und *ldpi* (low, < 120 dpi). Das Smartphone *G2 Touch*, das das mobile Endgerät für die Applikation der vorliegenden Arbeit darstellt, verfügt allerdings über die Plattform Android 1.5, auf welcher diese verschiedenen Auflösungen der Endgeräte noch nicht berücksichtigt werden. In solchen Fällen wird von einer *mdpi*-Auflösung ausgegangen.

In Android werden Icons unterschieden in *Launcher*-, Menü-, Statuszeilen-, *Tab*-, Dialog- und *ListView*-Icons. Aus der Bezeichnung heraus wird bereits klar, wo diese Icons jeweils zum Einsatz kommen. Die nachfolgende Tabelle enthält die Dimensionen der Icons, die kurz zusammengefassten Richtlinien für deren Gestaltung und Bespielicons. Die Dimensionen ergeben sich aus dem für Icons freigehaltenen Platz vordefinierter Bildschirmelemente.

	Dimension	**Design-Guidelines**	**Beispiel**
Launcher-Icon	48 x 48 Pixel	- modern, einfach, ikonisch - taktil, matt und strukturiert - ab Android 2.0: Aufrissansicht und von oben beleuchtet - beschränkte Farbpalette (eine neutrale und eine Primärfarbe), hoher Kontrast - bestehend aus einer kleinen in einer größeren Form	
Menü-Icon	48 x 48 Pixel	- gleiche Farbpalette für alle Menü-Icons einer Applikation - abgerundete Ecken - Aufrissansicht, flache Wirkung - innerer Schatten, äußerer Schein, Füllgradient	
Statusbar-Icon	32 x 32 Pixel	- wie Menü-Icons, aber kleiner und kontrastreicher - blindgeprägte Wirkung	
Tab-Icon	32 x 32 Pixel	- wie Menü-Icons, aber ohne äußeren Schein - unselektierte *Tab*-Icons haben geringeren inneren Schatten als selektierte *Tab*-Icons	
Dialog-Icon	32 x 32 Pixel	- heller Gradient, innerer Schatten - hervorstechend auf dunklem Hintergrund - Aufrissansicht, flache Wirkung	

ListView-Icon	32 x 32 Pixel	- wie Dialog-Icons, aber mit innerem Schatten von Lichtquelle über dem Icon	

Tab. 4.6: Android-Icon-Typen, ihre Dimensionen, Guidelines und Beispiele (ANDROID DEVELOPERS)

Es wird empfohlen, Standard-Icons der Android-Plattform zu verwenden, weil sie dem Benutzer bereits vertraut sind und eigene Icons an dieses Design anzulehnen, da eine Anwendung so professioneller wirkt (ANDROID DEVELOPERS).

Allgemeine Richtlinien für Positionssignaturen für die Bildschirmausgabe wurden im Rahmen einer Studienarbeit am Institut für Kartographie erarbeitet. Wie in Kapitel 4.1.1 festgestellt wurde, hinterlässt ein modernes Design einer Applikation einen positiven Eindruck beim Benutzer, weshalb im Folgenden speziell die in der Studienarbeit genannten Aspekte einer modernen Gestaltung kurz wiedergegeben werden sollen.

Moderne Gestaltung von Icons zeichnet sich unter anderem dadurch aus, dass Highlights gesetzt werden. Beliebt sind hierbei sogenannte Highlight-Linien, die entweder innerhalb des Icons auftreten oder in Form einer Außenlinie oder eines Umrisses und dadurch einen dreidimensionalen Eindruck hervorrufen.

Da Modernität mit Neuem assoziiert wird, werden Icons oft mit Glanzeffekten versehen. Jedoch ist ein Trend zu verzeichnen, der sich auch in den Android-Guidelines niederschlägt, nämlich dass eine matte Gestaltung bevorzugt wird, mit der aber trotzdem ein dumpfer Glanz zu realisieren ist.

Ferner können Icons mit Strukturen versehen werden, die verschiedene Materialien simulieren. Weitere Merkmale moderner Gestaltung sind abgerundete Ecken und Farbverläufe, die ebenfalls eine gewisse Plastizität hervorrufen. Des Weiteren wird als modernes Gestaltungsmittel die Perspektivität genannt, die aber den Richtlinien von Android widerspricht und sich ab einer bestimmten Größe ohnehin nur noch schwierig realisieren lässt.

Neben der modernen Gestaltung wird auf das Verwenden von Transparenz und Schatten sowie Animationen (z.B. durch Rotation, Blinken, Vibrieren) verwiesen (ALEXANDER, 2010).

Weitere Guidelines ergeben sich aus der Evaluation bestehender mobiler Applikationen wie beispielsweise der des Apps *VisitFinland*, der im Rahmen des Projektes *Country Portal for Finnish Tourism* entstanden ist. Die Ergebnisse dieser Evaluation sollen im Folgenden kurz zusammengefasst werden.

Um dem Benutzer eine Reihe an alternativen Optionen zur Auswahl zu stellen, können sowohl Radio-Buttons als auch Listen mit Links verwendet werden. Im Fall der Radio-Buttons kann der Benutzer einen Eintrag auswählen, weitere durchlesen und bei Bedarf seine Auswahl ändern. Erst danach wird der Benutzer auf die gewählte Seite geleitet, indem er auf einen Button unter der Liste klickt. Eine Liste mit Links hingegen leitet sofort nach dem Klick auf einen Listeneintrag zur nächsten Seite weiter. Letztere Möglichkeit wird von Benutzern bevorzugt, da sie weniger Klicks erfordert und somit zeitsparender und weniger aufwendig ist.

Im Allgemeinen führen sehr detaillierte Informationen und Anweisungen zu einer langsameren Navigation auf einem mobilen Gerät. Daher wird vom Benutzer eine kompaktere Variante mit weniger Anweisungen, aber mehr Platz für anderweitige Vorab-Informationen bevorzugt.

Eine ähnliche Verknappung kann durch die Verwendung von Icons erreicht werden. Zwar verbrauchen textliche Informationen weniger Speicherkapazität, jedoch nehmen Icons weniger Platz auf dem Bildschirm in Anspruch, sind auffälliger und schneller zu interpretieren.

Des Weiteren wird Scrollen auf einer Bildschirmseite gegenüber Vor-/Zurück-Links bevorzugt (SCHNEIDER ET AL, 2010).

4.3 Eine mobile Applikation für touristische Aktivitäten in der Pilotregion Sächsische Schweiz

4.3.1 Der Wanderkalender der Sächsischen Zeitung

Die im Rahmen dieser Diplomarbeit zu konzipierende touristische Kartenapplikation soll an den Wanderkalender der Sächsischen Zeitung angelehnt sein. Jedes Jahr erarbeiten Studenten des Studiengangs Kartographie der Technischen Universität Dresden für diesen Kalender zwölf Wanderungen zu einem jährlich wechselnden Thema und dokumentieren diese in Karten, Beschreibungen und Bildern. Der Wanderkalender ist im Jahr 2011 bereits in der 8. Folge erschienen (EDITION SÄCHSISCHE ZEITUNG). Jede dieser in dem Kalender enthaltenen Wanderrouten ist auf vier Kalenderseiten aufgeteilt: die erste Seite stellt das eigentliche Kalenderblatt mit Kalendarium und einem Foto dar, die zweite Seite liefert Informationen über das Gebiet, durch das die Wanderung führt, auf der Folgeseite ist eine Karte der Wanderung zu finden und die vierte Kalenderseite enthält eine Beschreibung der Wanderroute, unterteilt in mehrere Etappen. In Anhang XIV befinden sich die Kalenderblätter der Wanderroute ‚Entlang des Steinbruchpfads Wehlen', welche die Autorin der vorliegenden Arbeit für den Wanderkalender des Jahres 2010 erarbeitet hat und die als Pilotroute für die zu konzipierende Applikation dienen soll.

An dieser Stelle soll zusammengetragen werden, was für Material der Wanderkalender liefert, das für die mobile Applikation verwendet werden kann:

- Infobox mit Eigenschaften der Wanderroute:
    - Wegstrecke
    - Anfahrt
    - Weglänge
    - Dauer
    - Höhenunterschied

- ausführliche Beschreibung der Wanderroute (in mehrere Etappen unterteilt)
- Karte
- Informationen zum Wandergebiet (in mehrere Abschnitte unterteilt)
- Bilder

Es bietet sich an, die oben beschriebene Teilung einer Wanderung innerhalb des Kalenders in Routenbeschreibung, Karte und Informationen zum Wandergebiet in der Applikation beizubehalten. Von dem Kalendarium wird dabei abgesehen.

4.3.2 Konzeption der Applikation

Als Konzeptionseinstieg stellt sich die Frage, wie der Zugang zu den Touren gestaltet werden kann. Die gängigste Art und Weise bei den in Kapitel 4.1 untersuchten Applikationen ist eine Auflistung aller Routen. Vier dieser Applikationen stellen Routenvorschläge vor und zeigen sie in einer Listenansicht auf, zwei davon zusätzlich in einer Karte. Dabei wird jedoch jede Tour nur als Positionssignatur visualisiert, der Verlauf ist nicht erkennbar (was aber möglicherweise auch im Kartenmaßstab begründet liegt) - ein Nachteil, der in der zu konzipierenden Applikation nach Möglichkeit nicht auftreten soll. Es ist vorteilhaft, wenn der Benutzer selber über die Präsentationsart der verfügbaren Touren entscheiden kann. Den Einstieg in die Applikation soll also eine Auswahl zwischen einer Auflistung aller Touren und einer Karte, in die alle Touren eingezeichnet sind, darstellen. Sowohl in der Übersichtskarte als auch in der Liste kann eine Tour ausgewählt werden, die dann in der oben genannten Dreiteilung angezeigt wird.

In Anhang XV ist die geplante touristische Applikation in Form eines Flussdiagramms dargestellt. Hierbei repräsentiert jedes Rechteck eine neue Bildschirmseite der Applikation. Durch die Überschrift wird deutlich, was zu sehen ist; darunter sind alle Interaktionen aufgelistet, die innerhalb dieser *Activity* möglich sind. Diese sind unterteilt in *Klick*, *Karte* und *Optionsmenü*. *Klick* enthält alle Interaktionen, die durch einen Klick auf ein Objekt, beispielsweise einen Button oder eine Signatur, eine Aktion auslösen oder eine neue Bildschirmseite aufrufen. Enthält eine Bildschirmseite eine Karte, sind alle Interaktionen, die mit dieser möglich sind, unter *Karte* aufgelistet. Die Menüeinträge des Optionsmenüs der jeweiligen Bildschirmseite sind unter *Optionsmenü* versammelt. Von einer Auflistung der Kontextmenüeinträge wird in Anhang XV abgesehen, da in dieser Darstellung nicht das Kontextmenü einer jeden *View* beschrieben werden kann. Des Weiteren sind in Anhang XV die Wege, auf welchen der Benutzer zu den einzelnen Bildschirmseiten gelangen kann, eingezeichnet; allerdings nur eine Richtung, da ein Zurückspringen jederzeit über die *Zurück*-Taste des mobilen Endgerätes möglich ist.

Wie bereits erwähnt, wird in der ersten *Activity* der Applikation dem Benutzer zur Wahl gestellt, alle Wanderrouten entweder in einer Karte dargestellt oder in einer Listenansicht aufgezählt zu bekommen. Die Listenansicht soll aus zweizeiligen Listeneinträgen bestehen. In der ersten Zeile steht der Name, in der zweiten Zeile kurze Informationen wie Weglänge und Dauer der Wanderung. Die Liste kann über das Optionsmenü entweder nach der ersten Zeile, also alphabetisch, oder nach der Weg-

länge sortiert werden. Ebenso kann eine Suchfunktion aufgerufen werden, die alle angebotenen Wanderrouten nach einem vom Benutzer eingegebenen Begriff durchsucht. Das Suchergebnis wird ebenfalls in Form einer Listenansicht mit zweizeiligen Einträgen präsentiert. Diese Suchfunktion soll auch bereits auf dem Startbildschirm sowie in der Übersichtskarte aller Touren verfügbar sein. Ein Klick auf eine Tour in der Übersichtskarte liefert die gleichen Auskünfte über eine Route wie ein Listeneintrag (Name, Weglänge, Dauer). Danach können weitere Touren angeklickt oder eine Tour aufgerufen werden. Diese wird in Form einer dreiteiligen TabHost-Ansicht präsentiert. Die *Tabs* enthalten die drei oben genannten Teile einer Wanderroute: Tourenbeschreibung, Karte und Informationen zum Wandergebiet. Der Aufbau des *Tab*-Inhaltes von Tourenbeschreibung und Gebietsinformation ähnelt sich: der Fließtext ist in mehrere Etappen bzw. Abschnitte unterteilt, von denen anfangs nur die Überschriften sichtbar sind, um so trotz der begrenzten Displaygröße einen Überblick über den gesamten Inhalt zu geben. Mit einem danebenstehenden Button kann der dazugehörige Fließtext ein- und auch wieder ausgeblendet werden. In die Fließtexte sollen Fotos eingebunden werden, welche Links zu den entsprechenden POIs in der Karte enthalten. Der dritte *Tab* enthält eine Karte der Wanderung. Diese kann genauso wie die Übersichtskarte gezoomt und verschoben werden. Ebenso können einzelne Objekte angeklickt und Informationen dazu aufgerufen werden, d.h. zu der entsprechenden Stelle im Fließtext gesprungen werden (insofern vorhanden). Des Weiteren kann die aktuelle Position des Benutzers in der Karte angezeigt und über das Optionsmenü eine Legende der Karte aufgerufen werden. Letzteres wurde von der Autorin in den untersuchten touristischen Applikationen vermisst. Zwar können POIs ausgewählt werden, um Erläuterungen dazu zu erhalten, aber eine Erklärung der Signaturierung der Basiskarte war in keiner der Anwendungen vorhanden.

Als Basiskarte für diese Anwendung soll zunächst *OpenStreetMap* dienen. Standardmäßig kann in Android-Applikationen *Google Maps* eingebunden werden, jedoch ist diese Grundlage für einen Wander-App nur wenig geeignet, da keine kleinen Wege, sondern nur Straßen enthalten sind. In *OpenStreetMap*-Karten sind hingegen neben kleineren Wegen auch Klettergipfel, Höhlen, Steinbrüche, Parkplätze etc. eingezeichnet. Für die Einbindung von *OpenStreetMap* in Anroid-Anwendungen existieren quelloffene Frameworks. Ein solcher soll für die Wander-Applikation der Sächsischen Schweiz Anwendung finden. Kartographisch gesehen wäre die auf *OpenStreetMap*-Karten basierende *Reit- und Wanderkarte* (http://www.wanderreitkarte.de/) noch geeigneter, da sie - wie der Name bereits sagt - zusätzlich Reit- und Wanderwege, Höhenlinien und eine Schummerung sowie für Wanderkarten typische Objekte wie Wegemarkierungen, Rastplätze, Aussichtspunkte etc. enthält und die Beschaffenheit kleiner Wege sehr differenziert unterscheidet (Teerweg, Betonspurweg, Feldweg, Wiesenweg, Bergpfad etc.). Für diese Karte existiert allerdings noch kein Framework für die Einbindung in eine Android-Applikation. Ferner besagt die Lizenz der Reit- und Wanderkarte, dass es sich um ein privates Projekt handelt und alle Verwendungen der Karte kostenlos zugänglich sein müssen sowie dass der Server nicht als *Tileserver* vorgesehen und es demnach nicht erwünscht ist, den Server ohne Absprache als Datenquelle in ein Offline-Kartenprogramm oder Downloadscript einzubauen. Jedoch kann für größere Datenmengen ein Downloadzugang abonniert werden. Aus diesen Gründen soll zunächst auf *OpenStreetMap* ausgewichen werden.

4.3.3 Umsetzung der Konzeption

4.3.3.1 Autorenwerkzeuge: NetBeans IDE und Eclipse IDE

Bei der NetBeans IDE handelt es sich um eine Entwicklungsumgebung, die auf der Programmiersprache Java basiert und auf der NetBeans-Plattform läuft. Hauptsächlich wurde die NetBeans IDE für die Programmiersprache Java entwickelt, unterstützt aber des Weiteren auch C, C++ und dynamische Programmiersprachen. Ferner existieren neben *Plugins* sogenannte *Packs*, durch welche die IDE um weitere Funktionsmöglichkeiten in Form von Modulen und Bibliotheken erweitert werden kann (PETRI, 2008).

Eine Möglichkeit, mit der NetBeans IDE eine Android-Applikation zu entwickeln, besteht darin, das *NetBeans Mobility Pack* zu integrieren. Dieses erlaubt ein *Rapid Prototyping* von Anwendungen durch das *Drag- & Drop*-Prinzip. Die zweite Möglichkeit ist das Einbinden des *Android-Plugins* für NetBeans. Die Autorin der vorliegenden Arbeit hat sich für die zweite Möglichkeit, d.h. für das ‚händische Programmieren', entschieden. Die mobile touristische Anwendung für die Sächsische Schweiz wurde also mit der NetBeans IDE 6.8 und dem Android SDK 1.5 entwickelt.

Das Android SDK erlaubt es, die entwickelte Anwendung am PC zu testen. Dafür muss ein virtuelles Gerät (AVD) mit den Eigenschaften des Zielgerätes definiert werden, welches in Form eines *Emulators* simuliert wird. Abbildung 4.5 zeigt einen Screenshot dieses *Emulators*, welcher auch über NetBeans gestartet werden kann.

In der Endphase der Programmierarbeiten ist die Autorin dieser Arbeit jedoch auf die Eclipse IDE als Entwicklungsumgebung umgestiegen, da es bei der Entwicklung der Kartendarstellung im App Komplikationen mit NetBeans gab. Ebenso wie NetBeans handelt es sich bei der Eclipse IDE um eine quelloffene Entwicklungsumgebung. Ursprünglich wurde Eclipse ausschließlich als Entwicklungswerkzeug für die Programmiersprache Java genutzt, kann aber inzwischen auf Grund seiner Erweiterbarkeit ebenso für andere Aufgaben verwendet werden. Diese Erweiterungen sind teilweise quelloffen, teilweise proprietär und von kommerziellen Anbietern (SEEBOERGER-WEICHSELBAUM, 2006).

Für die Entwicklung von Android-Applikationen mit Eclipse muss das *ADT-Plugin* installiert werden, dann kann ebenfalls mit Eclipse das Android SDK genutzt werden. Auch Eclipse ermöglicht das Gestalten von Bildschirmseiten per *Drag & Drop*, jedoch ist die Autorin auch hier wieder über das ‚händische' Programmieren vorgegangen. Ebenso über Eclipse kann das entwickelte Android-Projekt emuliert werden. Von der Autorin wurde Eclipse in der Version 3.6 (Eclipse Helios) verwendet.

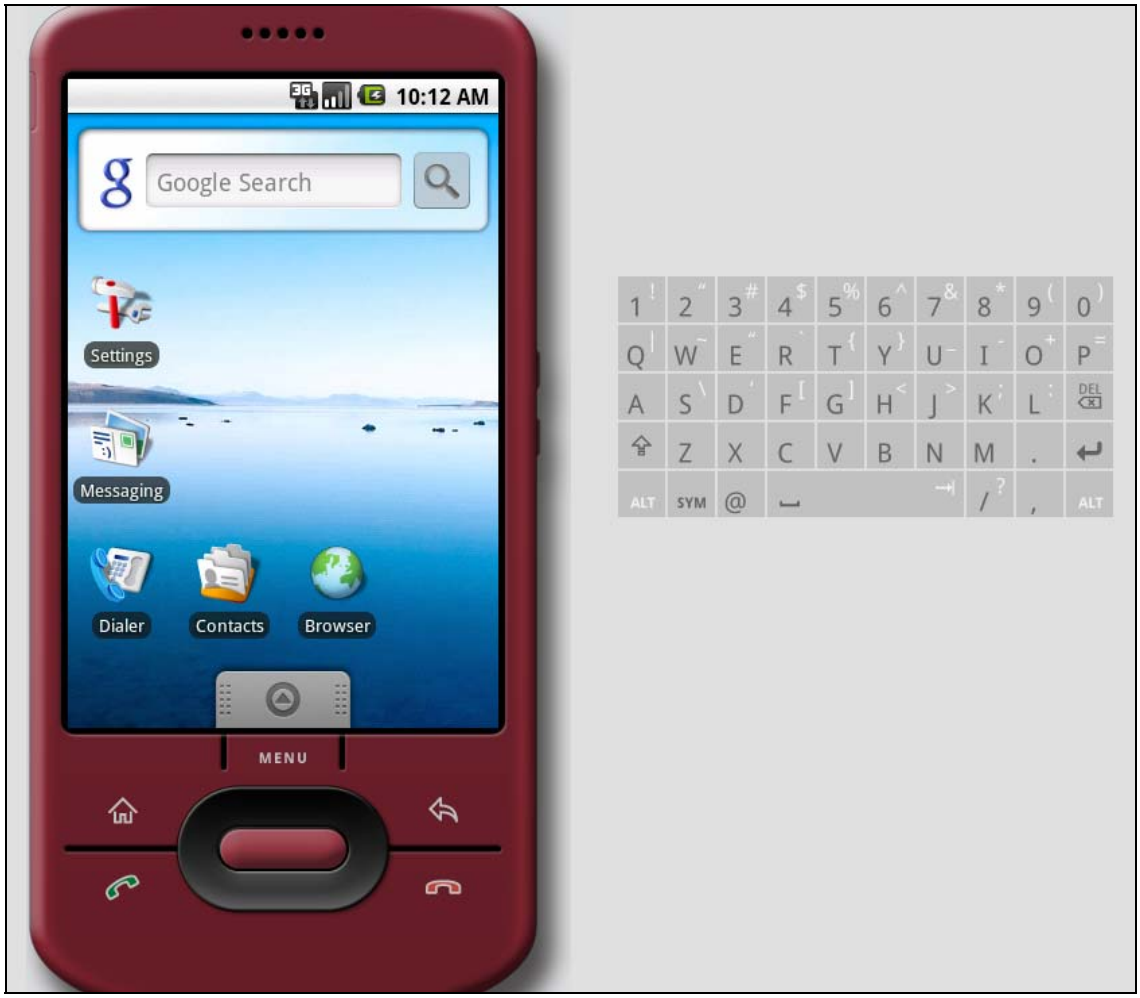


Abb. 4.5: Emulator des Android SDK 1.5

Die der Diplomarbeit angefügte CD-ROM enthält alle notwendigen Dateien für das Installieren von Eclipse als Entwicklungsumgebung für Android-Projekte sowie eine Installationsanleitung.

4.3.3.2 Programmiertechnische Umsetzung

Im Folgenden soll nun auf die praktische Umsetzung der Konzeption eingegangen werden. Anhang XVI zeigt wieder ein Flussdiagramm der mobilen Applikation ähnlich Anhang XV, aber diesmal sind für jede Bildschirmseite der Name der entsprechenden *Activity* eingetragen, ein Screenshot eingefügt und die Art des Android-Layouts genannt, auf dem die jeweilige Bildschirmseite basiert.

Den Einstieg in die Applikation ‚Wandern in der Sächsischen Schweiz' stellt ein *Splash Screen*, also ein Startbildschirm dar, der so lange angezeigt wird, wie die Applikation lädt (`SplashScreen`). Das `LinearLayout` des *Splash Screens* zeigt dasselbe Hintergrundbild der Sächsischen Schweiz wie die als nächstes aufgerufene Bildschirmseite `Startseite`, welche zusätzlich eine Überschrift mit dem Namen der Anwendung und zwei Button zeigt. Über das Optionsmenü kann die Suchfunktion aufgerufen oder die Applikation beendet werden. Je nachdem auf welchen Button geklickt wird, wird entweder `TourenListe` oder `TourenKarte` als nächste Bildschirmseite gestartet. Bei `TourenLis-`

te handelt es sich um eine Liste, die alle vorhandenen Wandertouren aufführt und auf TwoLine-ListItem, einer *View*, die innerhalb des ListView-Layouts verwendet wird, basiert. Jeder Listeneintrag besteht aus zwei Zeilen. In der ersten steht der Name, in der zweiten Weglänge und Dauer der Wanderung. Das Optionsmenü stellt zur Auswahl, die Liste entweder nach Weglänge bzw. alphabetisch nach Namen zu sortieren, die Suchfunktion zu nutzen oder die *Activity* zu beenden. Der Klick auf einen Listeneintrag ruft die *Activity* Tour auf. Bei dieser Bildschirmseite handelt es sich um einen TabHost, der drei TabWidgets beherbergt, welche die angestrebte Dreiteilung einer Wandertour realisieren. Im ersten TabWidget befindet sich die Beschreibung der Route (Tour_Route), im zweiten eine Karte (Tour_Karte) und im dritten weitere Informationen zum jeweiligen Wandergebiet (Tour_Info). Der erste *Tab* ist hierbei als Standard eingestellt, d.h. wird Tour für die gewählte Route erstmals aufgerufen, ist eingangs der *Tab* ‚Route' aktiviert.

Die Datenhaltung bezüglich aller Informationen zu den Wandertouren wurde durch die Autorin zunächst in Form von *Arrays* vorgenommen, die innerhalb einer XML-Datei im res-Ordner der Android-Applikation abgelegt wurden. Jedoch erwies sich zum einen die Datenabfrage und das Sortieren der Daten als relativ umständlich, zum anderen erlaubt die starre Struktur von XML nur ebenso umständlich ein Hinzufügen von neuen Daten. Aus diesen Gründen ist die Autorin im Laufe der Programmierarbeiten auf eine Datenhaltung in Form einer Datenbank umgestiegen. Hierbei wird das Datenbanksystem SQLite genutzt, welches im Lieferumfang von Android inbegriffen ist und einen effizienten Zugriff auf Datenbanken erlaubt. SQLite ist ein kompaktes, quelloffenes und relationales Datenbanksystem, das für den Einsatz auf mobilen Plattformen optimiert wurde. Es benötigt nur geringfügigen Speicherplatz und unterstützt nahezu alle Funktionen von Verwandten auf komplexen Serversystemen, wie die SQL-Syntax oder Transaktionen. Von jenen Verwandten unterscheidet sich SQLite darin, dass keine Installation und keine zentrale Konfiguration erforderlich ist sowie Zugriffe aus mehreren Anwendungen auf eine Datenbank erlaubt sind. Ebenso verfügt SQLite über dynamische Feldlängen, d.h. vordefinierte Feldlängen werden ignoriert und nur die tatsächliche Länge eines Feldes wird zum Speichern verwendet. Dadurch kann vorhandener Speicherplatz optimal genutzt werden (BECKER/PANT, 2009).

Die für die mobile Wander-Applikation der Sächsischen Schweiz verwendete Datenbank wurde mit dem Tool *SQLite Database Browser* erstellt, welches über ein GUI das Erzeugen und Editieren von SQLite-Datenbanken erlaubt und sich auf der dieser Arbeit angefügten CD-ROM befindet. Die Datenbank für die Wander-Applikation der Sächsischen Schweiz befindet sich im Ordner assets des Android-Projektes und heisst wanderfuehrer_db.db. Diese Datenbank besteht lediglich aus einer Tabelle mit dem Namen Touren. Pro Wanderroute gibt es einen Datensatz in der Tabelle. Der Aufbau dieser Tabelle mit Feldnamen und -typen ist in Anhang XVII ersichtlich. Das erste Feld der Tabelle ist vom Typ INTEGER PRIMARY KEY und stellt die ID eines jeden Datensatzes der Tabelle dar, ist somit also zwingend erforderlich. Das nächste Feld enthält den Namen der entsprechenden Wanderroute und ist demzufolge vom Typ TEXT. Die darauf folgenden fünf Tabellenfelder entsprechen den Einträgen der bereits oben genannten Infobox aus dem Wanderkalender: Weglänge, Dauer, Wegstrecke, Anfahrt und Höhenunterschied. Enthält eines dieser Felder eine Zahl, ist es vom Typ NUMERIC, andernfalls wieder vom Typ TEXT. Da jener Zahl stets eine Einheit zugeordnet werden kann, ist diese dem Feldnamen angefügt (z.B. _km oder _h). Alle Tabellenfelder, deren Name mit

Etappe_ beginnen, halten Daten für die *Activity* Tour_Route bereit, Tabellenfelder beginnend mit Info_ sind hingegen für Tour_Info bestimmt. Das Feld KML enthält Informationen, die für die Darstellung der Route in der Karte benötigt werden.

Wird die mobile Wander-Applikation der Sächsischen Schweiz zum ersten Mal aufgerufen, wird innerhalb der *Activity* Startseite die Datenbank aus dem assets-Ordner in das Verzeichnis data/data/org.me.wanderfuehrer/databases des mobilen Endgerätes kopiert. Auf dieses Verzeichnis greift die Applikation zu, um Daten aus der Datenbank abzurufen. Dies geschieht beispielsweise beim Erstellen der Listenansicht in TourenListe. Zunächst wird unter anderem eine Datenbank vom Typ SQLiteDatabase und ein Cursor erzeugt:

```
protected SQLiteDatabase db;
final Cursor dbCursor;
private static final String dbTable = "Touren";
boolean sortByAlphabet = true;
```

Diese Variablen werden benötigt, um die Datenbank zu öffnen und Daten auszulesen.

```
try {

      db = DbHelper.getDataBase();
      if (sortByAlphabet == true)
      {
            dbCursor = ListOrderedByName();
      }
      else
      {
            dbCursor = ListOrderedByDistance();
      }

      int NameColumn = dbCursor.getColumnIndex("Tour_Name");
      int WeglaengeColumn = dbCursor.getColumnIndex("Weglaenge_km");
      int DauerColumn = dbCursor.getColumnIndex("Dauer_h");

      int len = dbCursor.getCount();
      final String Touren [][] = new String [len][2];
      String Tourname;
      String Kurzinfo;

      startManagingCursor(dbCursor);

      dbCursor.moveToFirst();

      for (int i = 0; i < len; i++) {
            Tourname = dbCursor.getString(NameColumn);
            Kurzinfo = dbCursor.getFloat(WeglaengeColumn) + " km, "
                        + dbCursor.getString(DauerColumn) + " h";
            Touren[i][0] = Tourname;
            Touren[i][1] = Kurzinfo;
            dbCursor.moveToNext();
      }

      TourArrayAdapter adap = new TourArrayAdapter(this,
      R.layout.touren_liste, Touren);
```

```
	setListAdapter(adap);
}
finally
{
	if(db != null) {
		DbHelper.close();
	}
}
```

Funktionen wie `getDataBase`, `close` und weitere, die für die Verwaltung der Datenbank erforderlich sind, sind in der Java-Klasse `DataBaseHelper` zusammengefasst. Der Code dieser Klasse ist in Anhang XVIII zu finden. Mit der Funktion `getDataBase` wird die Datenbank geöffnet. Nun wird zunächst geprüft, ob die Variable `sortByAlphabet` vom Typ `Boolean` `true` ist. Bei Aufruf der *Activity* `TourenListe` wird sie anfangs als wahr deklariert; im Weiteren wird der Wert über das Optionsmenü festgelegt. Ist `sortByAlphabet` wahr, wird dort der Menüpunkt ‚Nach Weglänge sortieren' angeboten. Bei Auswahl dessen wird `sortByAlphabet` auf `false` gesetzt, woraufhin bei Neuaufruf des Optionsmenüs dem Benutzer ‚Alphabetisch sortieren' zur Auswahl steht. Ist `sortByAlphabet` wahr, wird die Funktion `ListOrderedByName` aufgerufen:

```
protected Cursor ListOrderedByName () {
	Cursor dbCursor = db.rawQuery("SELECT Tour_Name, Weglaenge_km,
					Dauer_h FROM " + dbTable + "
					ORDER BY Tour_Name ASC", null);
	return dbCursor;
}
```

Andernfalls erfolgt ein Aufruf der Funktion `ListOrderedByDistance`:

```
protected Cursor ListOrderedByDistance () {
	Cursor dbCursor = db.rawQuery("SELECT Tour_Name, Weglaenge_km,
					Dauer_h FROM " + dbTable + "
					ORDER BY Weglaenge_km ASC", null);
	return dbCursor;
}
```

In beiden Funktionen werden über einen SQL-Befehl die Felder `Tour_Name`, `Weglaenge_km` und `Dauer_h` aus der Datenbank-Tabelle `Touren` selektiert und entsprechend nach dem Namen der Wanderung bzw. nach der Weglänge sortiert. Der Inhalt all dieser ausgewählten Spalten wird mit einer `for`-Schleife in einen zweidimensionalen *Array* geschrieben, dessen Inhalt wiederum an den selbstimplementierten Typ `CustomArrayAdapter` übergeben wird, um anschließend in eine zweizeilige Listenansicht übertragen zu werden. Das Layout letzterer ist in der XML-Datei `touren_liste.xml` abgelegt. `CustomArrayAdapter` bedient jene selbstdeklarierte zweizeilige Listenansicht aus `touren_liste.xml`, da eine solche in dieser Form nicht standardmäßig in Android zur Verfügung steht. Schlussendlich wird die Datenbank wieder geschlossen. Der gesamte Quellcode von `TourenListe` befindet sich in Anhang XIX.

Ähnlich funktioniert die Suchfunktion der Applikation. Hier werden allerdings in der dazugehörigen SQL-Abfrage sämtliche Inhalte der Datenbank (abgesehen von dem Feld KML) mit dem vom Benutzer eingegeben Suchbegriff verglichen; im Suchergebnis hingegen, welches nur Datensätze mit einer Übereinstimmung enthält und das ebenso auf der Listenansicht aus `touren_liste.xml` basiert,

erscheinen wieder nur die Inhalte der Datenbankfelder `Tour_Name`, `Weglaenge_km` und `Dauer_h`. Ein solcher SQL-Befehl, der die Suchergebnisse alphabetisch nach dem Namen der Tour sortiert, lautet:

```
SELECT Tour_Name, Weglaenge_km, Dauer_h FROM dbTable
WHERE Tour_Name LIKE '%TourenListe.searchKeywords%'
OR Weglaenge_km LIKE '%TourenListe.searchKeywords%'
OR Dauer_h LIKE '%TourenListe.searchKeywords%'
OR Wegstrecke LIKE '%TourenListe.searchKeywords%'
OR Anfahrt LIKE '%TourenListe.searchKeywords%'
OR Hoehenunterschied_m LIKE '%TourenListe.searchKeywords%'
OR Etappe_1_Name LIKE '%TourenListe.searchKeywords%'
OR Etappe_1_Dauer_min LIKE '%TourenListe.searchKeywords%'
OR Etappe_1_Text LIKE '%TourenListe.searchKeywords%'
OR Etappe_2_Name LIKE '%TourenListe.searchKeywords%'
OR Etappe_2_Dauer_min LIKE '%TourenListe.searchKeywords%'
OR Etappe_2_Text LIKE '%TourenListe.searchKeywords%'
OR Etappe_3_Name LIKE '%TourenListe.searchKeywords%'
OR Etappe_3_Dauer_min LIKE '%TourenListe.searchKeywords%'
OR Etappe_3_Text LIKE '%TourenListe.searchKeywords%'
OR Etappe_4_Name LIKE '%TourenListe.searchKeywords%'
OR Etappe_4_Dauer_min LIKE '%TourenListe.searchKeywords%'
OR Etappe_4_Text LIKE '%TourenListe.searchKeywords%'
OR Info_1_Name LIKE '%TourenListe.searchKeywords%'
OR Info_1_Text LIKE '%TourenListe.searchKeywords%'
OR Info_2_Name LIKE '%TourenListe.searchKeywords%'
OR Info_2_Text LIKE '%TourenListe.searchKeywords%'
OR Info_3_Name LIKE '%TourenListe.searchKeywords%'
OR Info_3_Text LIKE '%TourenListe.searchKeywords%'
OR Info_4_Name LIKE '%TourenListe.searchKeywords%'
OR Info_4_Text LIKE '%TourenListe.searchKeywords%'
ORDER BY Tour_Name ASC
```

Wesentlich komplexer gestaltete sich das Füllen des Layouts der einzelnen *Tabs* der *Activity* `Tour` mit Inhalten aus der Datenbank. Für den *Tab* ‚Route' sind neben den Inhalten für die Infobox sämtliche Tabellenfelder vorgesehen, die mit der Bezeichnung `Etappe_` beginnen, denn wie bereits erwähnt, soll eine Wanderroute in mehrere Etappen unterteilt werden. Die Layout-Datei `tour_route.xml` stellt derzeit Platz für vier solcher Etappen bereit, kann aber jederzeit erweitert werden. Eine Etappe setzt sich aus einer Überschrift, in der die Wanderdauer dieser Etappe enthalten ist, aus Fließtext und einem Bild zusammen, wobei der Platz für das Bild nicht zwingend in Anspruch genommen werden muss. Dementsprechend sind die Felder der Tabelle benannt: `Etappe_1_Name`, `Etappe_1_Dauer_min`, `Etappe_1_Text`, `Etappe_1_Bild`, `Etappe_2_Name` etc.

Die schon erwähnte Layout-Datei, die für diese Daten vorgesehen ist, stellt eine Verschachtelung aus verschiedenen Android-Basislayouts dar, die in Anhang XX schematisch dargestellt ist. Kursiv gedruckte Wörter geben den ID-Namen der entsprechenden *View* an insofern vorhanden. Den Rahmen bildet ein `ScrollView`, welcher ein `RelativeLayout` umgibt. Dieses enthält an erster Stelle ein `TableLayout` mit einem weiteren `RelativeLayouts`. `TableLayouts` werden hier als Rahmen verwendet, da es kein entsprechendes Attribut für Android-*Views* gibt, um eine Linie um ein Objekt herum anzuzeigen. An dieser Stelle stellt das `TableLayout` die Infobox dar. Diese Infobox enthält fünf `RelativeLayouts`, die jeweils die Einfassung für zwei `TextViews` bilden. Die linke `TextView`

enthält einen der fünf Fakten, die in der Infobox genannt werden (Wegstrecke, Anfahrt, Weglänge, Dauer, Höhenunterschied); in die rechte `TextView` werden die entsprechenden Inhalte aus der Datenbank geschrieben. Daraufhin folgen vier `RelativeLayouts` mit identischem Aufbau für jeweils eine Etappe der Tour. Jede Etappe enthält zwei weitere `RelativeLayouts`. Das erste umgibt die Überschrift und einen Button, der für das Ein- und Ausblenden des zweiten `RelativeLayouts` verantwortlich ist, welches aus einer `TextView` für den Fließtext und einem `TableLayout` mit einer `ImageView` für ein Bild besteht. Die Verwendung der zahlreichen `RelativeLayouts` erscheint zunächst umständlich, ist jedoch erforderlich, um die gewünschte Anordnung der *Views* zu erzielen. Der entsprechende XML-Code dieses Layouts ist in Anhang XXI nachzulesen.

Der komplette Quellode aus `Tour_Route`, der das Füllen dieses Layout-Gerüstes mit Daten realisiert, befindet sich in Anhang XXII und soll im Folgenden erläutert werden. Die Vorgehensweise findet innerhalb einer `for`-Schleife statt, die von i = 0 bis 8 läuft. Dies entspricht der Nummerierung der IDs in der Layout-Datei `tour_route.xml`. Hat der Zähler i einen Wert zwischen 0 und 4, handelt es sich um einen Eintrag der Infobox. Zunächst wird eine Variable erzeugt, die den Namen der ID aus der XML-Layout-Datei beinhaltet:

```
String i_string = (new Integer(i)).toString();
int id_infobox = getResources().getIdentifier("route_infobox_" + i_string,
"id", getPackageName());
```

Über diese Variable wird auf die entsprechende `TextView` in der Infobox zugegriffen:

```
TextView infobox = (TextView) findViewById(id_infobox);
```

Diese `TextView` wird anschließend mit Inhalt gefüllt, beispielsweise:

```
infobox.setText(dbCursor.getString(dbCursor.getColumnIndex("Wegstrecke")));
```

Wobei:

```
dbCursor = db.rawQuery("SELECT * FROM " dbTable " WHERE Tour_Name =
'" TourenListe.sel_tour "'", null);
```

Folglich werden anfangs aus der Datenbank alle Felder jenes Datensatzes selektiert, bei dem der Inhalt des Feldes `Tour_Name` mit dem Namen der in `TourenListe` ausgewählten Wanderroute übereinstimmt.

Ist i größer als 4, handelt es sich nicht mehr um Einträge der Infobox, sondern um ein `RelativeLayout` einer Etappe, das mit Daten gefüllt werden soll. Da beispielsweise das Datenbankfeld `Etappe_1_Name` dem `RelativeLayout` mit der ID `route_etappe_5` entspricht, muss zunächst ein weiterer Zähler aus der Differenz dieser Bezeichner definiert werden:

```
String column_nr = (new Integer(i-4)).toString();
```

Enthalten die entsprechenden Felder in der Datenbank Daten, wird das zuvor unsichtbare `RelativeLayout` `route_rel_layout_`i auf sichtbar gesetzt. Im Folgenden werden ähnlich wie bereits oben beschrieben Variablen für die IDs aller enthaltenen *Views* definiert, um diese mit Inhalten zu versehen, wobei `route_rel_layout_hidden_`i vorerst unsichtbar bleibt und erst

eingeblendet wird, wenn ein Klick auf den Button neben der Etappenüberschrift erfolgt. Dieser stellt einen Expander dar und ändert sein Aussehen entsprechend der Sichtbarkeit von `route_rel_layout_hidden_i`:

```
if (route_rel_layout_hidden.getVisibility() == View.GONE) {
      route_rel_layout_hidden.setVisibility(View.VISIBLE);
      route_btn.setBackgroundResource(R.drawable.btn_expander_up);
}
else {
      route_rel_layout_hidden.setVisibility(View.GONE);
      route_btn.setBackgroundResource(
      R.drawable.btn_expander_down);
}
```

Ebenso wird geprüft, ob für die jeweilige Etappe ein Bild in dem Datensatz enthalten ist. Ist dies nicht der Fall, wird das die `ImageView` umgebende `TableLayout` unsichtbar geschaltet, da sonst der für das Bild vorgesehene Rahmen bzw. das `TableLayout` noch als kleines schwarzes Quadrat sichtbar ist.

Das Vorgehen für den *Tab* ‚Info' soll an dieser Stelle nicht erläutert werden, da es mit dem eben beschriebenen nahezu identisch ist, abgesehen davon, dass die enthaltenen Hintergrundinformationen zum Wandergebiet über keine Infobox verfügen, was den Quellcode verkürzt.

Im zweiten *Tab* der *Activity* `Tour` wird eine Karte angezeigt, in die der Verlauf der Wandertour sowie POIs entlang der Route eingezeichnet sind. Als Kartengrundlage wird *OpenStreetMap* verwendet, welches über eine Jar-Datei mit Namen *osmdroid* eingebunden wurde. *osmdroid* stellt Tools und *Views* zur Interaktion mit *OpenStreetMap* zur Verfügung, welche fast vollständig die *Google Maps* anzeigende standardmäßige `MapView` von Android ersetzen. *osmdroid* ist derzeit in der Version 3.0.3 verfügbar, bei der Programmierung der Applikation wurde *osmdroid* 3.0.2 verwendet. (OSMDROID).

Die für das Einzeichnen von Routenverlauf und POIs erforderlichen Daten sind in der Datenbank im Feld `KML` abgelegt und stellen den Inhalt einer KML-Datei dar, die während des Begehens der Wanderroute aufgezeichnet und deren Inhalt in die Datenbank kopiert wurde. Die Auszeichnungssprache KML befolgt die XML-Syntax und modelliert sämtliche Vektorobjekte wie Punkte, Polygone oder Linien als `Placemark`-Elemente. Deren Koordinaten werden stets im ‚Enkel'-Element `coordinates` abgelegt. Stellt dessen Elternelement einen *Tag* mit Namen `Point` dar, handelt es sich um eine Koordinate, die einen Punkt definiert; heißt das Elternelement hingegen `LineString`, sind im *Tag* `coordinates` mehrere Koordinatenpaare abgelegt, die ein Polygon oder einen Pfad bilden. Neben den Koordinaten können `Placemark`-Elemente einen Namen, eine Beschreibung, einen vordefinierten Stil, Betrachtungswinkel und -höhe, einen Zeitstempel sowie weitere Daten umfassen (KML TUTORIAL, 2011). Auf eben jene *Tags* `Point` und `LineString`, also auf alle POIs und den Routenverlauf, wird in `Tour_Karte` zugegriffen und der Inhalt des jeweiligen Kindelements `coordinates` in einen `String` eingelesen. Dieser `String` wird nach Kommas aufgesplittet, um so die einzelnen Koordinatenwerte zu erhalten:

```
String[] LngLatPoint = pair.split(",");
```

Da es sich im Falle des LineString-Elements um mehrere Koordinatenpaare handelt, erfolgt hier vorher eine weitere Aufsplittung nach Leerzeichen:

```
String [] pairs = path.split(" ");
for(int i=0;i<pairs.length;i++)
{
     String [] LngLatLineString = pairs[i].split(",");
}
```

Die LineString-Koordinaten werden anschließend durchlaufen und mit Hilfe der selbstimplementierten Klasse CustomOverlay in der Karte visualisiert: das jeweils erste und letzte Koordinatenpaar wird als Kreis eingezeichnet, um so Start- und Endpunkt der Route zu verdeutlichen, alle anderen Koordinaten werden der Reihe nach durch Linien miteinander verbunden. Der Quellcode der Klasse CustomOverlay ist in Anhang XXIII zu finden.

Bei den Point-Elementen werden auch die Kinder name und description eingelesen (sofern vorhanden) und nebst den Werten den Elementes coordinates an einen selbstimplementierten *Overlay* der Klasse CustomItemizedOverlayWithFocus übergeben, welcher am entsprechenden Ort in der Karte eine Positionssignatur einzeichnet und auf einen Klick hin Name und Beschreibung dieses POIs anzeigt. Verfügt ein POI über kein name-Kind, wird er ignoriert.

Das mehrfach erwähnte Zugreifen auf Knoten und Einlesen derer Inhalte geschieht durch Klassen des Packages javax.xml.parsers, die das Prozessieren von XML-Dokumenten ermöglichen. So können Knotenlisten, Knoten und Elemente erzeugt und auf deren Inhalte zugegriffen werden. Das Einlesen der Koordinaten aus dem coordinates-*Tag* eines LineString-Elementes aus der Datenbank heraus geschieht folgendermaßen:

```
Document doc = null;

final Cursor dbCursor = db.rawQuery("SELECT KML FROM " + dbTable + " WHERE
Tour_Name = '" + TourenListe.sel_tour + "'", null);
startManagingCursor(dbCursor);
dbCursor.moveToFirst();

byte[] b = dbCursor.getBlob(dbCursor.getColumnIndex("KML"));
InputStream is = new ByteArrayInputStream(b);

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder dbb = dbf.newDocumentBuilder();
doc = dbb.parse(is);
doc.getDocumentElement().normalize();

NodeList LineStringNodeList = doc.getElementsByTagName("LineString");
Node LineStringNode = LineStringNodeList.item(0);

Element LineStringElement = (Element) LineStringNode;
NodeList CoordList = LineStringElement.getElementsByTagName("coordinates");

Element CoordElement = (Element) CoordList.item(0);
String path = CoordElement.getFirstChild().getNodeValue();
```

Nach dem Parsen der KML-Inhalte wird zunächst eine Liste aller `LineString`-Knoten erzeugt. Im obigen Code wird im Folgenden auf den ersten Knoten dieser Liste zugegriffen, ein neues Element und daraus wiederum eine neue Knotenliste aller `coordinates`-Knoten erzeugt. Diese Liste enthält jedoch nur einen Knoten, da jedes `Placemark`-Element und damit jedes `LineString`- bzw. auch `Point`-Element nur über ein `coordinates`-Kind verfügen darf. Dieser Knoten wird wieder in ein Element verwandelt, von welchem der enthaltene Knotenwert in einen String eingelesen wird.

Sämtliche eingelesene Koordinatenpunkte werden zusätzlich in einen *Array* `bbList` geschrieben, aus dem anschließend die Boundingbox dieser Punkte berechnet wird, um so nach dem Ladevorgang ein gezieltes Heranzoomen an die Wanderroute umzusetzen:

```
BoundingBoxE6 bb = BoundingBoxE6.fromGeoPoints(bbList);
this.mapController.zoomToSpan(bb);
this.mapController.setCenter(bb.getCenter());
```

Während des Einlesens aller Koordinaten, Namen und Beschreibungen aus der Datenbank, wird ein an das Design der Applikation angepasster `ProgressDialog` der Klasse `CustomProgressDialog` angezeigt.

Im oberen Bildschirmteil der *Activity* `Tour_Karte` sind drei Buttons platziert. Mit dem linken und rechten kann in die Karte rein- bzw. rausgezoomt werden, der mittlere erlaubt das Aktivieren bzw. Deaktivieren der GPS-Funktion. Die aktuelle Position des Benutzers wird über die Klasse `CustomSimpleLocationOverlay` in Form einer Positionssignatur visualisiert, die der Karte als *Overlay* hinzugefügt wird:

```
CustomSimpleLocationOverlay curPosOverlay =
new CustomSimpleLocationOverlay (Tour_Karte.this);
Tour_Karte.this.mapView.getOverlays().add(curPosOverlay);
```

Das Starten der Lokalisierung erfolgt über

```
Tour_Karte.this.locationManager.requestLocationUpdates(LocationManager.GPS_
PROVIDER, 5000, 100, Tour_Karte.this);
```

Hiermit wird festgelegt, dass die Position des Benutzers aller 5000 ms bzw. aller 100 m, die er zurückgelegt hat, neu abgefragt wird. Ist dies der Fall, wird die Funktion `onLocationChanged` aufgerufen:

```
public void onLocationChanged(Location location) {

      if (location != null) {
            GeoPoint curPos = new GeoPoint (location.getLatitude(),
            location.getLongitude());
            Tour_Karte.this.curPosOverlay.setLocation(curPos);
            Tour_Karte.this.mapController.animateTo(curPos);
      }
}
```

Innerhalb dieser Funktion wird die Karte so verschoben, dass die aktuelle Position des Benutzers den Kartenmittelpunkt darstellt und an dieser Stelle wird die oben erwähnte Positionssignatur abgebildet.

Der komplette Quellcode der *Activity* `Tour_Karte` ist in Anhang XXIV zu finden. Dieser ähnelt dem Code für die Übersichtskarte aller Wanderrouten in `TourenKarte`, welche von der Startseite aus aufgerufen werden kann. Hier werden jedoch nur die Koordinaten aus den `LineString`-*Tags* aller Routen eingelesen; die POIs einer entlang einer Route werden nicht angezeigt. Zusätzlich wird am Startpunkt der Tour ein Marker (ebenfalls vom Typ `CustomItemizedOverlayWithFocus`) angezeigt, der auf einen Klick hin den Namen der Tour, Weglänge und Dauer anzeigt. In dieser Übersichtskarte kann eine Route ausgewählt werden, die anschließend wie bereits beschrieben in der *Activity* `Tour` präsentiert wird. Ferner sind die möglichen Interaktionen mit der Karte dieselben: Verschieben, Zoomen und Benutzerlokalisierung.

Sämtliche in der Android-Applikation enthaltenen Java-Klassen, deren Name mit `Custom` beginnen, sind von der Autorin eigens definierte Klassen, die bereits existierende Java-Klassen überschreiben und erweitern bzw. modifizieren. Der Name der originalen Klassen geht aus der Benennung hervor. So adaptiert beispielsweise `CustomDialog` die Java-Klasse `Dialog`. Gründe für das Kreieren solcher eigenen Klassen sind fehlende bzw. nicht zweckmäßige Funktionen der originalen Klasse oder ein nötiges Anpassen des Designs an jenes der Wander-Applikation der Sächsischen Schweiz.

Das komplette Android-Projekt der Wander-Applikation der Sächsischen Schweiz mit sämtlichem Quellcode und allen Ressourcen befindet auf der angefügten CD-ROM.

4.3.3.3 Design der zu entwickelnden Applikation

Für die Wander-Applikation der Sächsischen Schweiz wurde nicht das voreingestellte Standard-Design von Android, das von den Farben Schwarz, Grau und Orange geprägt ist, übernommen, sondern es wurde ein individuell angepasstes Design umgesetzt. Die Leitfarben, die dieses Design bestimmen, sind ein helles Gelb sowie Grün in verschiedenen Abstufungen. Das helle Gelb dient als Hintergrund, Grün wird verwendet für Schaltflächen, Balken, *Tabs*, Überschriften und ähnliches. Die unterschiedlichen Abstufungen von Grün kommen zum Einsatz, um die verschiedenen Zustände (normal, fokussiert, gedrückt) von beispielsweise Buttons oder Listeneinträgen zu unterscheiden (vgl. Abbildung 4.7) und um Farbverläufe für *Views* zu erzeugen. Die Wahl dieser Farben hat einerseits praktische, aber auch symbolhaltige Gründe. Gelb als heller Hintergrund wurde gewählt, da davon auszugehen ist, dass der Benutzer die Applikation im Freien und bei Tageslicht verwenden wird. Von daher ist ein heller Hintergrund mit dunkler Schrift günstig, da diese Variante auf einem Display im Hellen besser erkennbar ist als helle Schrift auf dunklem Grund. Die Symbolhaltigkeit der Farben sieht die Autorin in der Assoziation von Grün mit Natur und Gelb mit Sandstein, was durch die mobile Anwendung ‚Wandern in der Sächsischen Schweiz' gleichwohl bedient wird. In der folgenden Tabelle 4.7 sind alle in der entwickelten Applikation definierten Farben aufgeführt.

Farbe	Name	Hexadezimalwert	RGB
	Hintergrund	#FFFFE6	255, 255, 230
	Dunkelgrün	#0A6400	10, 100, 0
	Dunkeloliv	#577E43	87, 126, 67
	Olivgrün	#8BA169	139, 161, 105
	Hellgrün	#E0E7B8	224, 231, 184
	Schwarz	#000000	0, 0, 0
	Weiß	#FFFFFF	255, 255, 255
	Hellgrau	#EEEEEE	238, 238, 238

Tab. 4.7: In der Applikation verwendete Farben

Übernimmt man das Standard-Design von Android, würde beispielsweise `TourenListe` ein Erscheinungsbild haben wie links in Abbildung 4.6 zu sehen. Der rechte Bildteil zeigt das für die in dieser Arbeit zu entwickelnde Applikation angepasste Design.

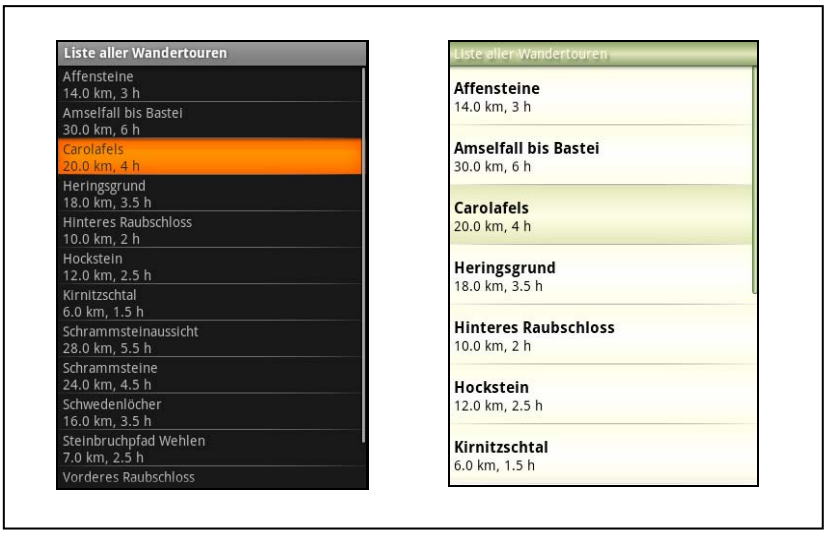


Abb. 4.6: Vergleich des Android-Standard-Designs einer ListView (links) mit einem angepassten Design (rechts)

Das Erscheinungsbild einiger *Views* kann recht leicht durch die schon erwähnten *Styles* und *Themes* geändert werden, ebenso beispielsweise der Hintergrund der einzelnen Listeneinträge in der obigen

Abbildung. Hierfür wird nicht eine einzelne Farbe verwendet, sondern ein Gradient, der einen Farbverlauf erzeugt. Gradienten werden in XML-Dateien definiert und im Verzeichnis `res/drawable` abgelegt. Der Gradient für den Hintergrund eines fokussierten Listeneintrages wird folgendermaßen deklariert:

```
<shape
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:shape="rectangle">
      <gradient
            android:startColor="@color/Hellgruen"
            android:centerColor="@color/Hintergrund"
            android:endColor="@color/Hellgruen"
            android:angle="-90" />
</shape>
```

Die hierbei verwendeten Farben sind in der Datei `color.xml` im Verzeichnis `res/values` als

```
<resources>
      <color name="Hintergrund">#FFFFE6</color>
      ...
      <color name="Hellgruen">#E0E7B8</color>
      ...
</resources>
```

definiert.

Eine Änderung des Designs anderer *Views* gestaltete sich dagegen deutlich umständlicher. Zum Beispiel wurde in der Wander-Applikation der Sächsischen Schweiz das Aussehen der *Tabs* in `Tour` modifiziert. Hierfür wird als Hintergrund eine Liste zweier Ebenen (ein farbiges Rechteck und ein Rechteck mit abgerundeten Ecken, gefüllt mit einem Gradienten und umgeben von einer Linie (siehe Anhang XXV)) verwendet; ein einfaches Zuweisen eines Gradienten als Hintergrund, so wie bei der Listenansicht erfolgt, war hier nicht möglich. Des Weiteren sollten in dieser Applikation nicht die Standard-Android-Buttons Verwendung finden, jedoch erhält man durch das Festlegen einer anderen Hintergrundfarbe für einen Standard-Button lediglich ein farbiges Rechteck. An dieser Stelle kommen die bereits erwähnten *NinePatch*-Grafiken zum Einsatz. Für die mobile Wander-Applikation der Sächsischen Schweiz wurden drei *NinePatch*-Dateien erzeugt, um eine angepasste Button-Darstellung zu erreichen. Diese sind in Abbildung 4.7 zu sehen.

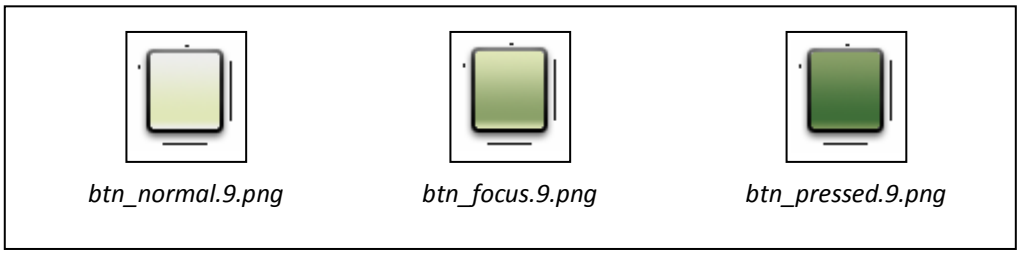


Abb. 4.7: NinePatch-Grafiken für eine angepasste Button-Darstellung

Jede Datei ist für einen von drei Button-Stadien (normal, fokussiert, gedrückt) vorgesehen. Die Zuordnung geschieht über einen sogenannten *Selektor*, welcher als Button-Hintergrund angegeben wird. *Selektoren* werden in Android stets verwendet, um verschiedene Zustände von *Views* durch verschiedene Farben oder Hintergründe zu visualisieren.

```
<selector
      xmlns:android="http://schemas.android.com/apk/res/android">
      <item android:state_focused="true"
            android:state_pressed="false"
            android:drawable="@drawable/btn_focus" />
      <item android:state_focused="true"
            android:state_pressed="true"
            android:drawable="@drawable/btn_pressed" />
      <item android:state_focused="false"
            android:state_pressed="true"
            android:drawable="@drawable/btn_pressed" />
      <item android:drawable="@drawable/btn_normal" />
</selector>
```

Des Weiteren wurden auf Grund des vom Android-Standard abweichenden Designs der Wander-Applikation eigene Dialogfenster definiert, unter anderem ein Suchfenster in Form der Java-Klasse `CustomSearchDialog`. Das Ergebnis im Vergleich zum standardmäßigen `SearchManager` ist in Abbildung 4.8 zu sehen.

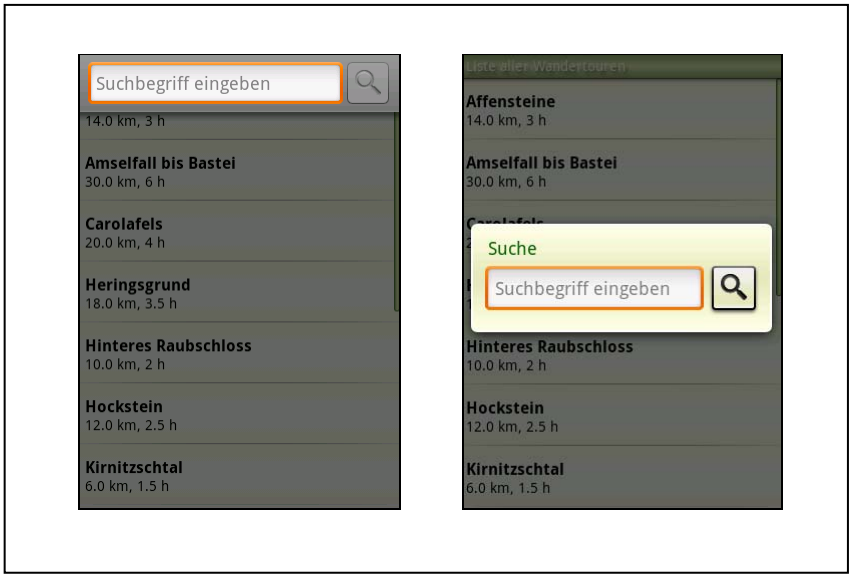


Abb. 4.8: Standard-Suchfeld von Android (links) und angepasstes Suchfenster (rechts)

Hier zeichnet sich bereits eine Schwachstelle von Android ab: das umständliche, teilweise undurchsichtige und dadurch langwierige Abändern und Anpassen des Design einer Applikation. Undurchsichtig, da die Vorgehensweisen von *View* zu *View* stark variieren. Konnte bei einigen oben beschriebenen Beispielen das Aussehen über Attribute der *View* (wie z.B. `android:background`)

modifiziert werden, musste für das Anpassen des Optionsmenü-Hintergrundes eigens eine Funktion definiert werden, die zu Beginn einer *Activity* aufgerufen wird:

```
protected void setMenuBackground(){

      Log.d(TAG, "Entering setMenuBackGround");
      getLayoutInflater().setFactory( new Factory() {

            @Override
            public View onCreateView ( String name, Context context,
            AttributeSet attrs ) {

                  if ( name.equalsIgnoreCase(
                  "com.android.internal.view.menu.IconMenuItemView" ) ) {
                        try {
                              LayoutInflater f = getLayoutInflater();
                              final View view = f.createView( name, null,
                              attrs );
                              new Handler().post( new Runnable() {
                              public void run () {
                                    view.setBackgroundResource(
                                    R.drawable.menue_bg_color);
                              }
                        } );
                       return view;
                       }
                  catch ( InflateException e ) {}
                  catch ( ClassNotFoundException e ) {}
                  }
            return null;
            }
     });
}
```

Die Unterschiede zwischen dem Standard-Optionsmenü und einem angepassten sind in Abbildung 4.9 ersichtlich.

Abb. 4.9: Android-Standard-Optionsmenü (links)
und ein farblich angepasstes Optionsmenü (rechts)

Wie in den oben stehenden Abbildungen erkennbar ist, wurde für den Hintergrund von Buttons, Listen- und Menüeinträgen mit vertikalen Farbverläufen gearbeitet, wodurch ein Glanzeffekt und eine gewisse Plastizität hervorgerufen wird, was den in Kapitel 4.2.3 genannten Aspekten eines modernen Designs entspricht. Weitere Guidelines aus diesem Kapitel wurden berücksichtigt, indem bei einer Reihe an alternativen Optionen keine Radio-Buttons, sondern eine Liste verwendet wurde ebenso wie das Scrollen auf Seiten, die den Bildschirm überschreiten, Anwendung findet und kein Vor- und Zurückblättern. Mehr zur Einhaltung von Design-Guidelines im folgenden Kapitel.

4.3.3.4 Icons der Applikation ‚Wandern in der Sächsischen Schweiz'

Das erste Icon, mit welchem der Benutzer bei der Verwendung einer Applikation in Berührung kommt, ist das *Launcher*-Icon. Durch dieses Icon wird die Anwendung zum einen in einer Auflistung aller auf dem Gerät installierten Applikationen oder auch auf dem *Home*-Bildschirm repräsentiert und zum anderen wird sie durch einen Klick auf dieses Icon gestartet. Somit sollte das *Launcher*-Icon die Applikation als Ganzes aussagekräftig verkörpern. Die Autorin hat für diesen Zweck versucht, ein Wahrzeichen der Sächsischen Schweiz für ein Abbilden auf lediglich 48 x 48 Pixeln zu stilisieren: die Bastei. Dieser Versuch scheiterte jedoch, woraufhin sich die Autorin am Logo des Nationalparks Sächsische Schweiz orientiert hat, welches den Lilienstein und die Elbe zeigt. Bei einer eventuellen Veröffentlichung der Applikation müssten hier allerdings die Urheberrechte geprüft werden. Zudem ist anzumerken, dass die Sächsische Schweiz in der Aufgabenstellung der Diplomarbeit als Pilotregion ausgeschrieben ist, also bei Ausweitung der Applikation auf zusätzliche Gebiete ohnehin ein anderes *Launcher*-Icon erforderlich wäre. Ergebnisse des Entwurfs von geeigneten *Launcher*-Icons durch die Autorin sind in Abbildung 4.10 zu sehen.

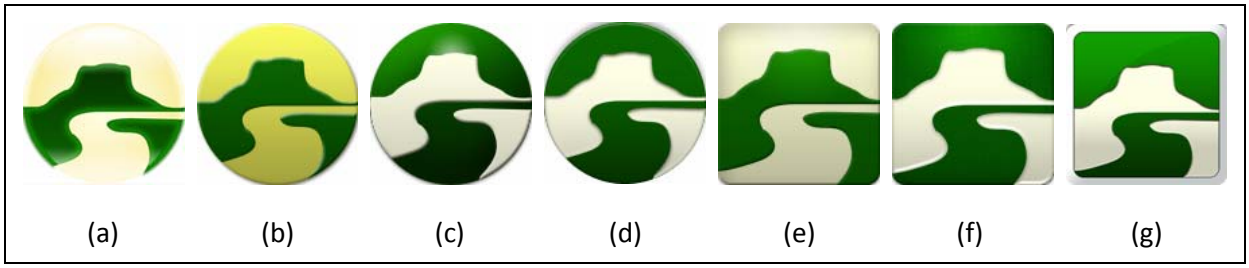


Abb. 4.10: Entwürfe für Launcher-Icons

Es ist erkennbar, dass während des Entwurfs mit verschiedenen Formen (rund (a, b, c, d) vs. eckig (e, f, g)), Hintergrundfarben (gelb (a, b, e) vs. grün (c, d, f, g) und inneren Schattenwürfen (hervortretende (a, b, c, e) vs. geprägte Wirkung des Logos (d, f, g)) experimentiert wurde. Die die Wander-Anwendung der Sächsischen Schweiz bestimmenden Farben Gelb und Grün treten bereits im *Launcher*-Icon auf. Dabei wurden die in Kapitel 4.2.3 genannten Aspekte der modernen Gestaltung einbezogen. In Variante g ist die erwähnte Highlight-Linie erkennbar. In nahezu jeder Variante wurde mit einem Glanzeffekt gearbeitet, wobei nur a und g eine scheinbar spiegelglatte Oberfläche haben, alle anderen Icons weisen einen matten Glanz auf. Auch in der Plastizität gibt es Unterschiede: a, c und g wirken auf den Betrachter sehr dreidimensional, wohingegen Variante d beinahe komplett flach erscheint. Ebenso wurde in jeder Variante mit einem Gradienten gearbeitet, entweder in der Hintergrundfarbe, der Vordergrundfarbe oder auch in beiden. Die Autorin hat sich schlussendlich für Variante f als *Launcher*-Icon entschieden, da sich einerseits die eckige Form gut in gängige Android-*Launcher*-Icons einreiht und andererseits durch verschiedene Schattenwürfe bzw. Lichteinfälle und einen inneren Schein ein geringer Glanz und eine nicht zu extreme Plastizität erzielt wird und diese Variante damit ein in den Augen der Autorin angenehmes Mittelmaß aller anderen bildet.

Bei Icons, die innerhalb der Applikation Anwendung finden, hat sich die Autorin weitestgehend an den Standard-Icons von Android 1.5 und *osmdroid* orientiert, um so dem Benutzer einen Wiedererkennungswert zu gewährleisten und damit die Bedienung zu erleichtern, hat die Icons jedoch an das Design des Apps angepasst.

An dieser Stelle sollte erwähnt werden, dass in der Regel so verfahren wird, dass Icons, die auf einem Button platziert werden, diesem über das Attribut `android:drawableTop` zugewiesen werden. Bis auf eine Ausnahme ist die Autorin jedoch so vorgegangen, dass für solche Buttons, die keine Schrift, sondern ein Icon zeigen, eigene PNG-Grafiken erstellt wurden. Hierbei handelt es sich ausschließlich um runde Buttons, da bei diesen in der Mehrzahl der Fälle das Abbilden von Icons über das XML-Attribut `android:drawableTop` auf die Button-Oberfläche die kreisrunde Form des Buttons zu einer Ellipse verzerrt. Mit einer eigenen Grafikdatei für jeden runden Button mit einem Icon wird dem vorgebeugt. Die eben erwähnte Ausnahme stellt ein rechteckiger Button dar, für den dieselbe *NinePatch*-Datei wie für Buttons mit Schrift verwendet wird. Das Skalieren kreisförmiger *NinePatch*-Dateien würde in nahezu jedem Fall zu einer Verzerrung führen.

Anhang XXV führt alle in der Applikation verwendeten Icons auf und vergleicht sie mit dem Standard von Android 1.5 bzw. *osmdroid*. Handelt es sich um einen Button, sind jeweils alle drei Stadien abgebildet, um so auch die farblichen Unterschiede zu verdeutlichen. Die Icons der *Tabs* sind sowohl in

ihrem selektierten als auch in ihrem unselektierten Aussehen aufgeführt. Bei der Gestaltung eigener Icons wurden die Guidelines des Android-SDK berücksichtigt. Die hierfür verwendete Software ist Adobe Photoshop.

4.3.3.5 Probleme mit Android 1.5

Die Aufgabenstellung besagt, dass eine mobile Kartenapplikation für das am Institut für Kartographie vorliegende Smartphone *G2 Touch* entstehen soll. Da dieses Gerät unter Android 1.5 läuft, musste die Programmierung der Anwendung ebenso auf dieser Plattform erfolgen. Dabei haben sich im Verlauf der Applikationsentwicklung einige Probleme aufgetan.

Die neueste Version ist Android 3.0 (API Level 11); Android 1.5 (API Level 3) ist also bereits relativ veraltet, was bei der Programmierung spürbar ins Gewicht fiel, da im API Level 3 einige Funktionen oder Attribute, die in der Android-Dokumentation vermerkt sind, noch nicht verfügbar waren. Das Gestalten eigener *Tabs* ist ein Beispiel dafür. Abbildung 4.11 zeigt das angepasste *Tab*-Design der entwickelten Applikation. Es ist zu erkennen, dass sich unter den nicht selektierten *Tabs* ein grauer Balken befindet anstatt der olivgrünen Linie, wie sie unter dem Tab ‚Route' zu sehen ist. Um diesen Balken auszublenden, ist das Attribut `setStripEnabled` vonnöten, welches allerdings erst seit Android 2.2 (API Level 8) existiert. Ein ausgefeiltes individuelles Design unter Android 1.5 ist demnach teilweise nicht bzw. nur mit Kompromissen möglich.

Abb. 4.11: Angepasstes Tab-Design unter Android 1.5

Weitere Probleme ergaben sich beim Einfügen von Bilddateien in die Datenbank. Dies kann nicht über das schon erwähnte Tool *SQLite Database Browser* vorgenommen werden, sondern muss programmiertechnisch erfolgen. Der dafür verwendete Quellcode ist in dem auf der CD-ROM angefügten Android-Projekt nicht enthalten, da dieses Hinzufügen im Vorhinein im Rahmen der Datenbankerstellung erfolgte. Ein exemplarischer Code folgt.

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
Bitmap bm = BitmapFactory.decodeResource(getResources(),
R.raw.steinbruchpfad_wehlen_etappe_1);
bm.compress(Bitmap.CompressFormat.PNG, 100, baos);

ContentValues cv = new ContentValues();
cv.put("Etappe_1_Bild", baos.toByteArray());

try {
      baos.close();
}
catch (IOException ex) {
```

```
}

db.insert("Touren", null, cv);
```

Die zweite Eingangsvariable der aufgerufenen Funktion `bm.compress(Bitmap.CompressFormat.PNG, 100, baos)` gibt den Komprimierungsfaktor an, mit dem die Bilddatei in die Datenbank übertragen wird. Dieser Wert muss zwischen 0 und 100 liegen, wobei 100 die höchste Qualität bedeutet. Da in diesem Fall das Zielformat PNG gewählt wurde, welches verlustfrei ist, sollte jeder beliebige Wert keinen Qualitätsverlust herbeiführen. Das Auslesen und Dekodieren der Bilddateien aus der Datenbank und das anschließende Visualisieren zeigten jedoch deutlich, dass ein solcher vorlag. Es ist nicht offiziell bestätigt, dass dies mit Android 1.5 zusammenhängt, jedoch zeigte der Versuch der Autorin, die Bilder unter Android 2.2 mit dem selben Code zur Datenbank hinzuzufügen, ein verlustfreies Ergebnis (siehe Abbildung 4.12), welches letztendlich für die bereitgestellte Datenbank verwendet wurde. Da dieses Hinzufügen wie bereits erwähnt nicht bei Aufruf der Applikation erfolgt, sondern im Rahmen der Datenbankerzeugung geschah, stellt es in diesem Zusammenhang kein Problem dar. Ein solches tritt erst auf, wenn die Anwendung dahingehend erweitert werden würde, dass der Benutzer selber Routen anlegen und somit auch unter Android 1.5 Bilder hochladen und zur Datenbank hinzufügen kann.

Abb. 4.12: Ausschnitt eines unter Android 1.5 (links) und Android 2.2 (rechts) der Datenbank hinzugefügten Bildes

Wie in der zweiten Zeile von Tabelle 4.6 anklingt, haben sich die Design-Guidelines für Android ab Android 2.0 verändert. Bei dem Entwurf der Icons für die Wander-Applikation der Sächsischen Schweiz hat sich die Autorin an den aktuellen Richtlinien orientiert, da sich eine Anwendung für Android 1.5 kaum halten wird.

Ein Problem, das nicht speziell das API Level 3, sondern Android allgemein betrifft, ist die Tatsache, dass Android als neue und einer raschen Entwicklung unterliegende Software-Plattform relativ schlecht dokumentiert ist. Zwar bietet die offizielle Homepage für Entwickler (http://developer.android.com/index.html) sehr detaillierte Referenzen über verfügbare *Packages* und Klassen mit allen Attributen und Methoden, aber nur einige einfache Anwendungsbeispiele und ein Tutorial für Einsteiger. Ein tiefergehendes und vor allem durchgängiges Tutorial sucht man ver-

geblich. Auch in der Literatur sind bisher nur wenige Bücher erschienen. Speziell für die bereits erwähnte komplizierte Gestaltung eines individuellen Designs ist eine intensive Suche nach Lösungen in Developer-Communities (wie z.B. *Android People* (http://www.androidpeople.com/), *anddev.org* (http://www.anddev.org/), *Stack Overflow* (http://stackoverflow.com/)) oder privaten Entwickler-Blogs (wie z.B. *Vidar Vestnes Blog* (http://vidarvestnes.blogspot.com/), *Sunny Dhillon Personal Blog* (http://www.sdhillon.com/)) erforderlich. Ebenso kann das Einsehen existierender quelloffener Apps hilfreich sein (*Project Hosting on Google Code* (http://code.google.com/hosting/)).

4.3.4 Derzeitiger Entwicklungsstand der Applikation

Der mit dieser Arbeit eingereichte Entwicklungsstand der Wander-Applikation der Sächsischen Schweiz entspricht weitestgehend der Konzeption aus Anhang XV. Die einzige *Activity*, die nicht implementiert wurde, ist die Legende für die zwei Kartendarstellungen. Des Weiteren fehlen die Verlinkungen von Texten bzw. Bildern und POIs in der Karte untereinander. Stattdessen enthalten die POIs der Karte im fokussierten Zustand Auszüge aus den Fließtexten der *Activities* `Tour_Route` bzw. `Tour_Info`. Ferner sind in der Applikation keinerlei Kontextmenüs implementiert, jedoch wurden solche in der Konzeption ohnehin nicht berücksichtigt.

Derzeit enthält nur die Pilotroute ‚Steinbruchpfad Wehlen' tatsächliche Daten aus dem Wanderkalender der Sächsischen Zeitung. Die Datenbankfelder aller anderen Wandertouren sind lediglich mit Platzhaltertexten gefüllt, über Bilddaten verfügen sie nicht. Die KML-Daten der Pilotroute entstammen einer Konvertierung der KMZ-Datei von der Homepage des Instituts für Kartographie. Diese enthielt jedoch nur den Verlauf der Wanderroute, die POIs wurden manuell durch die Autorin hinzugefügt. Um dennoch die Übersichtskarte in `TourenKarte` testen zu können, wurden für drei weitere der insgesamt 14 Wandertouren KMZ-Dateien von *EveryTrail* heruntergeladen und ebenso in KML-Dateien konvertiert. Hierfür wurde das Tool *RouteConverter* verwendet. Jene drei Routenverläufe sind ebenfalls nur als Platzhalter aufzufassen. Infolgedessen sei darüber hinweg zu sehen, dass die POIs der Routen und deren Beschreibungen in dieser Applikation nicht zweckmäßig sind bzw. ebenso die Namen sowie Weglängen- und Dauerangaben sämtlicher Platzhalterrouten nicht immer Sinn ergeben.

An dieser Stelle soll noch angemerkt werden, dass der Ladevorgang der KML-Daten aller Routen für `TourenKarte` sehr viel Zeit in Anspruch nimmt und die *Activity*, selbst nachdem dieser Vorgang abgeschlossen ist, nur sehr langsam reagiert. Dies liegt vermutlich daran, dass für jede Verbindungslinie zwischen zwei Punkten der Route sowie für jeden Start- und Endpunkt ein neuer *Overlay* zur Karte hinzugefügt wird. In der *Activity* `TourenKarte` sind dies beim derzeitigen Stand der Applikation 7337 Stück. Dadurch wird ganz offensichtlich der Arbeitsspeicher von zu vielen Daten in Anspruch genommen. Im Übrigen konnte die in Kapitel 4.3.3.2 erwähnte Funktion `zoomToSpan` zum gezielten Heranzoomen an eine Wanderroute in der Karte nur in `TourenKarte` verwendet werden. Auf Grund eines Bugs kann diese Funktion nicht innerhalb der `onCreate`-Methode zum Einsatz kommen, deswegen hat die Autorin sie in die `onWindowFocusChanged`-Methode geschrieben, die bei Aufruf einer *Activity* ohnehin ausgeführt wird. Dies funktioniert jedoch nicht bei `Tour_Karte`, weil es sich hier durch das *Tab*-Layout um eine *Activity* innnerhalb einer *Activity*

handelt, wodurch die `onWindowFocusChanged`-Methode bereits vor Einlesen der KML-Daten aufgerufen wurde. Da die Autorin im Zeitrahmen der Diplomarbeit keine Lösung für dieses Problem finden konnte, wird lediglich der Mittelpunkt der Kartendarstellung aus allen eingelesenen Punkten berechnet und anschließend ein festgelegtes Zoomlevel von 13 eingestellt. Für die vier vorliegenden Wanderrouten entsteht ein passender Kartenausschnitt, für weitere kann dies keinesfalls garantiert werden.

An dieser Stelle soll die Analyse, welche an den zu Beginn von Kapitel 4 beschriebenen neun mobilen Applikationen durchgeführt wurde, ebenso an der Wander-Applikation der Sächsischen Schweiz erfolgen. Auf eine genaue Erläuterung wird verzichtet, da die Funktionen dieser Anwendung bereits sehr detailliert dargestellt wurden, deswegen geschieht die Analyse lediglich in tabellarischer Form. Die Zahlen in der Kopfzeile der Tabelle entsprechen abermals der Nummerierung der in Kapitel 4.1.1 aufgeführten Kriterien.

	1.	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**	**10.**	**11.**	**12.**
Wandern in der Sächsischen Schweiz	✓	✓	–	–	–	–	–	–	✓	–	–	✓

Tab. 4.8.: Funktionsumfangs der mobilen Wander-Applikation der Sächsischen Schweiz

Der derzeitige Stand der entwickelten Applikation erfüllt vier der zwölf Kriterien, was 33% entspricht.

4.3.5 Ausblick

Da die Programmier- und Entwicklungsarbeiten an der zu entwickelnden Wander-Applikation der Sächsischen Schweiz bei Weitem nicht abgeschlossen sind, soll an dieser Stelle ein Ausblick auf nötige Fortentwicklungen bzw. mögliche Erweiterungen gegeben werden.

Nahe liegend ist, an die im vorherigen Kapitel genannten Lücken anzuknüpfen, also folglich tatsächliche Routen mit Daten zu erstellen, die Verlinkungen zwischen Texten bzw. Bildern und POIs in der Karte zu realisieren, eine Legende zu gestalten und Kontextmenüs zu erzeugen bzw. die schon vorhandenen Optionsmenüs zu erweitern. Letzteres könnte eine Hilfsfunktion zur Bedienung und Handhabung der Applikation umfassen sowie ein Einstellungsmenü, in dem der Benutzer beispielsweise verschiedene Farbschemata oder auch unterschiedliche Kartengrundlagen wählen kann (was auch über *osmdroid* möglich ist). Die Legende der Karten müsste jedoch dynamisch angelegt werden, da sich aus verschiedenen Zoomleveln auch verschiedene Zeichenerklärungen ergeben. Des Weiteren wäre es noch vorteilhaft, die Fotos der Wanderrouten mit Bildunterschriften zu versehen und vielleicht überdies pro Tour als vierten *Tab* eine Galerie mit allen Bildern zu implementieren, so dass die Bilder im Fliesstext nur als kleine Vorschau dargestellt werden. Ein weiteres Sortier-Kriterium für die Auflistung aller Wanderrouten in `TourenListe` bzw. `Suchergebnis` könnte die Entfernung des Benutzers zum Startpunkt der einzelnen Touren sein.

Eine fortschreitende Umsetzung des Wanderkalenders der Sächsischen Zeitung ist möglicherweise verbunden mit einem Überschreiten der Grenzen der Sächsischen Schweiz, da sich der Kalender nicht nur auf dieses Gebiet beschränkt und es ohnehin nur mit wenigen Routen abdeckt. Somit wären bei einem Hinzukommen neuer Wanderrouten und einer damit verbundenen Ausweitung des angebotenen Wandergebietes einerseits das *Launcher*-Icon und der Name der Applikation zu überdenken, andererseits wäre zu überlegen (auch wenn in der Konzeption anders vorgesehen), in der Übersichtskarte nur die Startpunkte der Touren zu zeigen, da der Verlauf ab einem bestimmten Zoomlevel ohnehin nicht mehr erkennbar ist und dies zugleich den Ladevorgang und sicher auch die Reaktionsgeschwindigkeit der *Activity* `TourenKarte` um ein Vielfaches beschleunigen, also generell die Performance verbessern würde. Ferner könnte die Suchfunktion dieser *Activity* derart gestaltet werden, dass das Ergebnis nicht eine Listenansicht aller Übereinstimmungen darstellt, sondern ebenfalls eine Kartendarstellung.

Die Aufgabenstellung dieser Arbeit spricht von einer Anlehnung an den Wanderkalender der Sächsischen Zeitung, daher hat sich die Autorin mit den der Applikation auf das Wandern beschränkt, jedoch wäre es ebenso möglich, es bei der Sächsischen Schweiz als räumlichen Geltungsbereich der Anwendung zu belassen und zukünftig ebenso weitere Sportarten einzubeziehen, wie beispielsweise Radfahren oder auch Klettern, wofür die Sächsische Schweiz schließlich zahlreiche Möglichkeiten bietet.

Eine weitere Option für die zukünftige Weiterentwicklung dieser Applikation wäre das Herstellen eigener Karten. Hierbei würde aber wahrscheinlich nur das jeweilige Wandergebiet mit eigenen Karten abgedeckt werden können, da eine Komplettabdeckung je nach Ausweitung der Applikation einen sehr hohen Aufwand bedeutet. Der Vorteil einer Applikations-eigenen Karte ist, dass dem Benutzer verschiedene Kartenstile zur Auswahl gestellt werden könnten und dass dieser nicht auf eine Internetverbindung angewiesen ist. Eine weitere Möglichkeit wäre, ab einer bestimmten Zoomstufe zwischen *OpenStreetMap* und der eigenen Karte zu wechseln.

Bei dem in diesem Kapitel bereits erwähnten Erstellen neuer Wanderrouten ist zu beachten, dass auf Dateien im `assets`-Verzeichnis, in dem ja die Datenbank der Applikation abgelegt ist, nur bis zu einer Größe von 1 MB zugegriffen werden kann, da sie bei einer Überschreitung dieses Grenzwertes komprimiert werden, was bei Android 1.5 Probleme hervorruft (wie beim Einlesen der Grafikdateien in die Datenbank deutlich wurde). Um dies zu umgehen, kann die Datei mit einer Endung versehen werden, die eine Komprimierung ausschließt, so wie beispielsweise `.png` oder `.mp3`. Wird allerdings die Datenbank und somit der gesamt App zu groß und nimmt zu viel Speicherkapazität auf dem Endgerät in Anspruch, sollte über eine Serverlösung oder einen Webservice nachgedacht werden. Die gegenwärtige Größe der Datenbankdatei beträgt 991 KB.

An der Oberflächengestaltung der Applikation könnte noch hinsichtlich der Bildschirmperspektive gearbeitet werden. Aus dem üblichen Halten eines Smartphones ergibt sich für das Display ein Hochformat; wird das Gerät jedoch um 90° gedreht, wechselt die Bildschirmperspektive automatisch ins Querformat. Für diesen Fall können in Android verschiedene Layouts geschaffen werden, welche anhand des Attributs `orientation` mit seinen möglichen Werten `portrait` (Hochformat) und `landscape` (Querformat) als Einschränkungskriterien eingesetzt werden. Derzeit ist das Layout

Wander-Applikation der Sächsischen Schweiz für `portrait` optimiert, dennoch leidet zumindest die Funktionalität nicht unter einem Wechsel der Perspektive ins Querformat.

Wie bereits in Kapitel 4.3.3.5 angeklungen ist, sollte das API Level der Wander-Applikation der Sächsischen Schweiz unbedingt auf einen aktuellen und gängigen Stand gebracht werden. Zwar läuft die Applikation auch auf Android-Plattformen höherer Versionen, dennoch könnten so die in Kapitel 4.3.3.5 geschilderten Probleme beseitigt werden. Ebenso vom Aspekt der Gängigkeit betroffen ist das Format, in dem die Daten über den Wanderroutenverlauf vorliegen. Die Autorin hat sich für KML entschieden, weil diese Auszeichnungssprache eine klare Gliederung und eine gute Einheitlichkeit aufweist, da es sich um einen Standard des Open Geospatial Consortium (OGC) handelt. Jedoch ist dieses Format nicht der gängige Output von Geräten mit GPS-Funktion, sondern dient der Beschreibung von Geodaten für die Client-Komponenten der Programme *Google Earth* und *Google Maps*. Hier bietet es sich an, auf das verbreitete Format GPX umzusteigen, da sich fast alle proprietären Standards in dieses offene Format umrechnen lassen (WIKIPEDIA). Bleibt es dennoch bei KML als Eingabeformat für den *Trail* der Wanderrouten, könnte für die Aufzeichnung der Android-App *RouteTracker* verwendet werden, der es erlaubt, Wegpunkte mit Notizen und einem Foto zu versehen und Routen im KML-Format auszugeben.

Während der Konzeption wurde von der Autorin dieser Arbeit in Erwägung gezogen, die Anwendung *Mobile Atlas Creator* in die Vorbereitung des Kartenmaterials für die Applikation einzubeziehen. *Mobile Atlas Creator* erlaubt es, offline-Karten in Form von Kacheln aus online-Quellen wie *Google Maps* oder *OpenStreetMap* zu erzeugen und somit den Benutzer von einer Internetverbindung unabhängig zu machen (MOBILE ATLAS CREATOR, 2008). Es bietet sich an, davon Gebrauch zu machen und nur Kacheln der benötigten Wandergebiete zur Verfügung zu stellen, jedoch wurden im März 2011 alle Downloads von *Mobile Atlas Creator* von den Entwicklern suspendiert (SOURCEFORGE, 2011). Im Falle einer zukünftigen Verfügbarkeit dieser Anwendung könnte sie allerdings doch noch Verwendung finden. Bis dahin kann man sich auf die Eigenschaft von *osmdroid* berufen, einmal angezeigte Kacheln auf dem Endgerät zu speichern und somit bei wiederholter Nutzung auch ohne Internetverbindung verfügbar zu machen.

5. Schlussfolgerungen

An konkreten Ergebnissen liefert die vorliegende Diplomarbeit ein Kommunikationsmodell der mobilen Kartographie und eine mobile Applikation auf Basis des Wanderkalenders der Sächsischen Schweiz. Das Kommunikationsmodell berücksichtigt und verbindet die im theoretischen Teil der Arbeit betrachteten Aspekte des Kontexts, der Adaption und der nutzergenerierten Inhalte. Mit dem abgeleiteten Kommunikationsmodell wurde der Versuch unternommen, die vorhandenen praktischen Möglichkeiten und die theoretischen Fundamentalregeln zu verbinden und die in der Motivation genannte Lücke zwischen technologischen Möglichkeiten und kartographischer Theorie etwas zu schließen, zumindest was den Bereich der mobilen Kartographie anbelangt.

Vergleicht man jedoch die im theoretischen Teil der Arbeit dargestellten Möglichkeiten mit dem Stand der in Kapitel 4.1 untersuchten mobilen Applikationen, zeichnet auch hier eine Lücke zwischen technologischen Möglichkeiten und kartographischer Theorie ab, allerdings eine konträre zu der von KRAAK und ORMELING (2010) wahrgenommenen. Die beschriebenen Vorgänge der Adaption und Personalisierung wurden zwar für Projekte wie *CRUMPET* (POSLAD ET AL, 2001), *USHER* (TITKOV/POSLAD, 2004) oder das formale Modell für mobile Kartenadaption von RAUBAL und PANOV (2009) konzipiert und teilweise auch prototypisch implementiert, doch finden solche Umsetzungen in eher alltäglichen Applikationen wie den im Rahmen dieser Arbeit untersuchten keine Anwendung. In diesem Hinblick kann die technologische Entwicklung mit den theoretischen Überlegungen nicht Schritt halten.

Inwiefern die konzipierte und prototypisch entwickelte Applikation diesen theoretischen Überlegungen Rechung trägt, soll an dieser Stelle geprüft werden. Der Kontext des Benutzers wird insofern berücksichtigt, dass die Applikation nicht versucht, eine möglichst große Anzahl an Nutzungsfällen abzudecken, sondern auf Benutzer abzielt, die in der Sächsischen Schweiz eine Wanderung unternehmen wollen. Somit wäre die Kontextdimension der Aktivität abgedeckt. Die Situation des Benutzers wird durch die in der Applikation angebotene Funktion der Nutzerlokalisierung berücksichtigt. Eine Benutzermodellierung findet nicht statt, was innerhalb der kurzen Nutzungsdauer ohnehin schwer zu realisieren ist. Es handelt sich bei der Wander-Applikation der Sächsischen Schweiz folglich nicht um eine adaptive, sondern um eine adaptierbare Anwendung. Eine Adaptivität liegt nur vor, wenn die Nutzerlokalisierung aktiviert ist und die Karte kontinuierlich auf den Standort des Benutzers zentriert wird. Die Nutzerlokalisierung kann vom Benutzer selbst ein- und ausgeschaltet werden und ist somit adaptierbar. Eine weitere Adaptierbarkeit liegt hinsichtlich verschiedener Aspekte vor: der Benutzer muss eine Route auswählen, die durch sein gewünschtes Wandergebiet führt, kann auf Wunsch Informationen einblenden (sowohl in der Karte als auch im Fließtext) und in der Karte den Ausschnitt und das Zoomlevel seinen Anforderungen entsprechend festlegen. Dem Benutzer werden also nicht die benötigten Informationen sofort präsentiert, sondern er muss sie selber auswählen bzw. bei Bedarf anzeigen lassen. Eine Berücksichtigung des mobilen Endgerätes findet momentan insofern statt, dass die Applikation für das in der Aufgabenstellung genannten Smartphone *G2 Touch* optimiert wurde, für weitere Geräte liegt eine solche nur in der Hinsicht vor, dass vor dem Starten der Nutzerlokalisierung geprüft wird, ob die GPS-Funktion am Gerät überhaupt aktiviert ist.

Die Vorschläge von MENG (2005) bezüglich der Designmethoden für egozentrische Karten wurden nur teilweise befolgt. So findet auf Wunsch des Nutzers eine Zentrierung der Karte auf den Benutzerstandort statt und die fokussierten POIs in der Karte stellen eine Form des von MENG (2005) vorgeschlagenen ‚Einzelfensters mit Details auf Anfrage' dar. Die weiteren in Kapitel 3.2.5 genannten Designmethoden der standortabhängigen bzw. verschachtelten LoDs und der Raumkontraktionen lassen sich im derzeitigen Zustand der Applikation ohnehin nur schwer realisieren, da kein eigenes Kartenmaterial verwendet, sondern auf *OpenStreetMap*-Daten zurückgegriffen wird.

In die Wander-Applikation der Sächsischen Schweiz werden momentan keine nutzergenerierten Inhalte eingebunden; im Zusammenhang mit dem Thema der vorliegenden Arbeit ist es dennoch nahe liegend, bei einer zukünftigen Weiterentwicklung *User Generated Content* in die Applikation einfließen zu lassen und dem Benutzer die Möglichkeit zu bieten, innerhalb der Anwendung eigene Routen zu erstellen und der Datenbank hinzuzufügen. Dafür ist eine *Activity* mit Texteingabefeldern entsprechend den Feldern in der Datenbank denkbar. Anschließend könnte ein Abgleich mit der ursprünglichen Datenbank auf einem Server erfolgen. Wurden von anderen Benutzern neue Wandertouren hinzugefügt, kann ein Update der Datenbank auf dem Smartphone angeboten werden, um dem Benutzer diese neuen Touren zugänglich zu machen. Da Android-Smartphones *Augmented Reality*-Anwendungen unterstützen, könnte ebenso innerhalb der Wander-Applikation der Sächsischen Schweiz davon verschiedentlich Gebrauch gemacht werden, wodurch eine weitere Adaptivität in die Applikation eingebunden werden würde. Einerseits könnte eine Navigation des Benutzers entlang der Wanderroute durch eine *Augmented Reality* erfolgen, andererseits wäre es auch möglich, mit Hilfe einer solchen Klettergipfel, Berg- und Felsnamen oder sogar Kletterwege an Felsen in der Umgebung anzuzeigen.

Die Konzeptions- und Entwicklungsarbeiten an der mobilen Wander-Applikation der Sächsischen Schweiz sowie das Ergebnis derer und mögliche zukünftige Erweiterungen spiegeln sich sehr gut in dem abgeleiteten kartographischen Kommunikationsmodell der mobilen Kartographie wieder, sind aber ebenso repräsentativ für die moderne Kartographie. Es wurde deutlich, dass sich der Kartograph - so wie im Kommunikationsmodell in Anhang X dargestellt - in Gestalter und Entwickler aufspalten lässt. Im Falle dieser Arbeit ist die Rolle des Entwicklers dominanter als die des Kartographen. Zwar besagt die Aufgabenstellung, dass das Ziel eine Kartenapplikation ist, jedoch war es notwendig mehr als nur eine reine Kartendarstellung zu implementieren. Die Kartendarstellung letztendlich basiert nicht auf eigenen Karten, sondern bis jetzt auf bereits vorhandenem Kartenmaterial von *OpenStreetMap*; lediglich eine Visualisierung der Wanderrouten und deren POIs in dieser Karte sowie des aktuellen Nutzerstandortes musste gestaltet werden. Dies spiegelt den gegenwärtigen Zustand der Kartographie wieder: technologische Entwicklungen führen zu einem starken ‚Technologiedruck' und kostenfreie bzw. *Open Source*-Lizenzmodelle zu einer verstärkten Nutzung vorhandenen Kartenmaterials (welches in diesem Falle wohlgemerkt aus nutzergenerierten Inhalten besteht) für eigene Zwecke. Mehr dazu im folgenden Kapitel. Wird die Wander-Applikation der Sächsischen Schweiz in Zukunft hinsichtlich nutzergenerierter Inhalte erweitert, zerfällt auch hier der Benutzer in Datenerzeuger und Datennutzer und wird damit zum *ProdUser*. Seine eigenen qualitativen Inhalte und Fotos werden dann in die Applikation eingebunden, und seine Rohdaten in Form von GPS-Tracks in der Karte entsprechend den Festlegungen des Kartographen visualisiert. Aus

allen in der Applikation vorhandenen Information erfolgt bei einer Nutzung eine Adaption an den Informations- und Interaktionsbedarf des Benutzers durch eine Adaptierbarkeit der Inhalte. Auch dies ist im Modell abgebildet. Diese Informationen fließen in die Welt des *ProdUsers* ein, indem sie Handlungen - in diesem Fall während der unternommenen Wanderung - zur Folge haben.

6. Diskussion

Was bereits in der Motivation der vorliegenden Arbeit angeklungen ist und sicher auch innerhalb der Arbeit wiederholt deutlich wurde, ist dass die Kartographie mehr denn je von technischen Entwicklungen und Impulsen vorangetrieben und beeinflusst wird. Wurde zur Zeit der Entstehung der kartographischen Kommunikationstheorie viel Wert auf Fragen der Semantik und Syntax gelegt, liegt gegenwärtig das Hauptaugenmerk auf der kontextuellen und pragmatischen Dimension.

Nicht nur, dass eine Karte nicht länger ein Artefakt ist, sogar das Grunddogma der Kartographie, nämlich lesbare graphische Darstellungen zu erzeugen, findet kaum Beachtung. Dadurch entstehen zwei Parallelwelten: die traditionelle Kartographie mit ihrer langen Geschichte, erprobten Methoden und profundem Wissen; daneben die zahlreichen Amateurnutzer, die mit den ihnen zur Verfügung stehenden Technologien auf eine ‚spielerische' und oft der Unterhaltung dienenden Art, private Kartendarstellungen erzeugen und individuelle räumliche Daten visualisieren.

Dennoch ist die Hauptaufgabe der Kartographie dieselbe geblieben: die Aufbereitung und Gestaltung von Präsentationen raumbezogener Sachverhalte und Objekte durch graphische Mittel mit dem Ziel, das Bedürfnis des Kartennutzers nach raumbezogenen Informationen möglichst effizient zu erfüllen. Auf diese Aufgabe haben natürlich die verfügbaren Technologien Einfluss. Und genau an dieser Stelle muss der Versuch, die zwei Parallelwelten einander wieder näher und vielleicht sogar in Einklang zu bringen, ansetzen. Einerseits ist die traditionelle Kartographie gefordert, ihre bewährten Standards in effiziente und einfach zu verwendende Werkzeuge zu gießen, um der fehlenden Qualität in kartographischen Darstellungen entgegenzuwirken [GARTNER/SCHMIDT, 2010]. Andererseits ist es unumgänglich, dass die Kartographie mit ihrer zunehmenden Interdisziplinarität über ihre Schwerpunktthemen nachdenkt. Dies ist bereits einmal geschehen, zwei Jahrzehnte nachdem die Kartographie trotz jahrhundertelanger Tradition erst 1949 von der UNO als offizielle Wissenschaft anerkannt wurde und damit ebenso ihren Standpunkt finden musste. In jener Zeit sind die ersten kartographischen Kommunikationsmodelle entstanden.

Der Sinn und Zweck solcher Modelle liegt darin, einen Gegenstandsbereich zu strukturieren, von unwichtigen Aspekten zu abstrahieren und somit Forschungsfragen zu formulieren [MALETZKE, 1998]. In diesem Fall dienten sie der Kartographie sicher eher für die Identifikation und Abgrenzung gegenüber anderen Wissenschaften. Zwar ist die Kartographie längst keine junge Wissenschaft mehr, die einer Selbstfindung bedarf, jedoch würde es ihr zugute kommen, erneut ihre Kernthemen zu überdenken und zu finden und damit ihren Aufgabenbereich neu zu definieren. Davon profitieren einerseits Forschung und Lehre, die sich ihrer Interdisziplinarität zwar bewusst sein müssen, durch eine klare Abgrenzung zu benachbarten Wissenschaften jedoch gestärkt werden würden, da sie sich wieder auf ihre gewachsenen Werte konzentrieren und berufen und diese zeitgemäß einbringen können. Andererseits und eben genau dadurch gewinnen auch kartographische Darstellungen als Ergebnis und Produkt der Kartographie an neuer Qualität, da auch die syntaktische und semantische Dimension der kartographischen Kommunikation wieder mehr ins Bewusstsein rücken. Damit ist nicht das Anstreben einer ‚Autarkie' der Kartographie gemeint, sondern eine Weiterentwicklung und das Wahrnehmen gegebener Chancen, durch die neue Brücken geschlagen werden können zu den

Nachbarwissenschaften wie beispielsweise der Informatik, auf die die Kartographie nach wie vor angewiesen ist, allerdings können Aufgaben neu verteilt und abgegeben werden.

Quellenverzeichnis

Literaturquellen

ALEXANDER, NICOLE: *Gestaltung von Positionssignaturen für die Bildschirmausgabe*. Studienarbeit. Institut für Kartographie, Technische Universität Dresden, 2010.

BECKER, ARNO / PANT, MARKUS: *Android. Grundlagen und Programmierung.* 1. Auflage. Heidelberg: dpunkt.verlag, 2009.

BOLLMANN, JÜRGEN / KOCH, WOLF GÜNTHER: *Lexikon der Kartographie und Geomatik: in zwei Bänden.* Heidelberg, Berlin: Spektrum Akademischer Verlag, 2001.

COLEMAN, DAVID J. / GEORGIADOU, YOLA / LABONTE, JEFF: *Volunteered Geographic Information: the nature and motivation of produsers*. In: International Journal of Spatial Data Infrastructures Research, Vol. 4. Brüssel: Joint Research Centre, 2009.

DEY, ANIND K. / ABOWD, GREGORY D.: *Towards a Better Understanding of Context and Context-Awareness.* In: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing. London: Springer, 1999.

FREITAG, ULRICH: *Kartographische Konzeptionen. Beiträge zur theoretischen und praktischen Kartographie. 1961 - 1991.* In: Berliner Geowissenschaftliche Abhandlungen, Reihe C, Kartographie, Band 13. Berlin: Selbstverlag Fachbereich Geowissenschaften, Freie Universität Berlin, 1992.

GARTNER, GEORG / SCHMIDT, MANUELA: *Moderne Kartographie - Technologische Entwicklungen und Implikationen*. In: Kartographische Nachrichten, 60. Jahrgang, Heft 6. Bonn: Kirschbaum Verlag, 2010.

GOODCHILD, MICHAEL F.: *Citizens as sensors: the world of volunteered geography.* In: GeoJournal, Vol. 69, No. 4. Dordrecht: Springer Verlag, 2007a.

GOODCHILD, MICHAEL F.: *Citizens as Voluntary Sensors: Spatial Data Infrastructure in the World of Web 2.0.* In: International Journal of Spatial Data Infrastructures Research , Vol. 2. Brüssel: Joint Research Centre, 2007b.

HAKE, GÜNTER / GRÜNREICH, DIETMAR: *Kartographie*. 7. Auflage. Berlin, New York: Walter de Gruyter Verlag, 1994.

HAKE, GÜNTER / GRÜNREICH, DIETMAR / MENG, LIQIU: *Kartographie*. *Visualisierung raum-zeitlicher Informationen.* 8. Auflage. Berlin, New York: Walter de Gruyter Verlag, 2002.

HEIDMANN, FRANK: *Aufgaben- und nutzerorientierte Unterstützung kartographischer Kommunikationsprozesse durch Arbeitsgraphik. Konzeptionen, Modellbildung und experimentelle Untersuchungen.* Dissertation. Fachbereich VI Geographie/Geowissenschaften, Universität Trier, 1999.

HOFFMANN, KARSTEN: *Nutzergenerierte Karten und kartographische Kommunikation im Web 2.0*. In: 58. Deutscher Kartographentag 2010 - Programmheft. Berlin, Potsdam: Deutsche Gesellschaft für Kartographie e.V., 2010.

HAKLAY, MORDECHAI / WEBER, PATRICK: *OpenStreetMap: User-Generated Street Maps.* In: IEEE Pervasive Computing, vol. 7, no. 4. Washington: IEEE Computer Society , 2008.

HÖLLDOBLER, TANJA: *Temporäre Benutzermodellierung für multimediale Produktpräsentationen im World Wide Web.* In: Europäische Hochschulschriften, Reihe XLI, Informatik. Frankfurt am Main: Peter Lang, Europäischer Verlag der Wissenschaften, 2001.

JÄGER, KAREN: *Mobile Dienste und ihr Potential für die Tourismusbranche: Eine nutzerorientierte Untersuchung am Fallbeispiel "Freizeitnavigator" des Tourismusverbandes Mecklenburg-Vorpommern e.V. - Kurzfassung.* Masterarbeit. Hochschule für nachhaltige Entwicklung Eberswalde, 2005.

KELNHOFER, FRITZ: *Thematische Kartographie zwischen Persistenz und Innovation*. Unveröffentlichter Vortrag zum Ehrenkolloquium Wolf Günther Koch am 15.01.2003, Technische Universität Dresden.

KOLÁČNÝ, A.: *Cartographic Information – A Fundamental Concept and Term in Modern Cartography.* In: The Cartographic Journal, Vol. 6 (1). London: Maney, 1969.

KOLBITSCH, JOSEF / MAURER, HERMANN: *The Transformation of the Web: How Emerging Communities Shape the Information we Consume.* In: Journal of Universal Computer Science, Jahrgang 12, Heft 2. Graz: Technische Universität Graz, 2006.

KRAAK, MENNO-JAN / ORMELING, FERJAN: *Cartography. Visualization of Geospatial Data*. 3. Auflage. Harlow: Pearson Education Limited, 2010.

KÜPPER, AXEL: *Location-based services: fundamentals and operation.* Chichester: John Wiley & Sons Ltd, 2005.

LECHTHALER, MIRJANKA / STRASSER, MARTINA / TODOR, RAZVAN: *Die Rolle der Kartographie in räumlichen Entscheidungsprozessen. The role of cartography in spatial decision processes.* In: Mitteilungen der Österreichischen Geographischen Gesellschaft, Band 149. Wien: Österreichische Geographische Gesellschaft, 2007.

MALETZKE, GERHARD: *Kommunikationswissenschaft im Überblick : Grundlagen, Probleme, Perspektiven.* Opladen, Wiesbaden; Westdeutscher Verlag, 1998.

MENG, LIQIU: *Ego centres of mobile users and egocentric map design.* In: Map-based Mobile Services. Theories, Methods and Implementations. Heidelberg: Springer Verlag, 2005.

MENG, LIQIU / REICHENBACHER, TUMASCH: *Map-based Mobile Services.* In: Map-based Mobile Services. Theories, Methods and Implementations. Heidelberg: Springer Verlag, 2005.

MENG, LIQIU: *Methoden zur Gestaltung egozentrischer Karten*. In: Kartographische Schriften, Band 8. Bonn: Kirschbaum Verlag, 2004.

MEYER, HANNES: *Benutzermodellierung in adaptiven Web-Anwendungen.* In: Advanced Web Technology – Effektive Bereitstellung von Information im World Wide Web. Berlin: Logos-Verlag, 2010.

MOSEMANN, HEIKO / KOSE, MATTHIAS: *Android. Anwendungen für das Handy-Betriebessystem erfolgreich programmieren*. München, Wien: Carl Hanser Verlag, 2009.

OGRISSEK, RUDI: *Theoretische Kartographie*. Gotha: VEB Herrmann Haack, Geographisch-Kartographische Anstalt Gotha, 1987 (Studienbücherei Kartographie, Band 1).

PETERSON, MICHAEL P.: *Interactive and Animated Cartography*. New Jersey: Prentice-Hall, 1995.

PETRI, JÜRGEN: *Netbeans RCP. Das Entwicklerheft*. 1. Auflage. Köln: O'Reilly Verlag, 2008.

POLCHAU, JOSIAS: *OpenStreetMap im Vergleich zu anderen Geodatenanbietern*. Seminararbeit. Fachhochschule Wedel, 2009.

POSLAD, STEFAN / LAAMANEN, HEIMO / MALAKA, RAINER / NICK, ACHIM / BUCKLE, PHIL / ZIPF, ALEXANDER: *CRUMPET: Creation of user-friendly mobile services personalized for tourism.* In: Proceedings of 3G 2001 – Second international conference on 3G mobile communication technologies. London: Institution of Engineering and Technology, 2001.

PRELL, KARL M.: *Informationswiedergabe in topographischen Karten: ein Beitrag zur theoretischen Fundierung kartographischer Ausdrucksformen unter besonderer Berücksichtigung der Siedlungsdarstellung.* Dissertation. Institut für Kartographie und Topographie, Universität Bonn, 1983.

QUIRING, OLIVER / SCHWEIGER, WOLFGANG: *Interaktivität – ten years after*. In: Medien & Kommunikationswissenschaft, 54. Jahrgang, Heft 1. Baden-Baden: Nomos Verlagsgesellschaft, 2006.

RAUBAL, MARTIN / PANOV, ILIJA: *A Formal Model for Mobile Map Adaptation*. In: Location Based Services and TeleCartography II. From Sensor Fusion to Context Models. Berlin, Heidelberg: Springer-Verlag, 2009.

REICHENBACHER, TUMASCH: *Adaptive egocentric maps for mobile users.* In: Map-based Mobile Services. Theories, Methods and Implementations. Heidelberg: Springer Verlag, 2005.

REICHENBACHER, TUMASCH: *Adaptive und kontextbezogene Visualisierung für mobile Nutzer.* In: Kartographie 2001 - multidisziplinär und multimedial: Beiträge zum 50. Deutschen Kartographentag. Berlin, Offenbach: Wichmann, 2001.

REICHENBACHER, TUMASCH: *Mobile Cartography - Adaptive Visualisation of Geographic Information on Mobile Devices.* Dissertation. Fakultät für Bauingenieur- und Vermessungswesen, Technische Universität München, 2003.

REICHENBACHER, TUMASCH / ANGSÜSSER, STEPHAN / MENG, LIQIU: *Mobile Kartographie - eine offene Diskussion.* In: Kartographische Nachrichten, 52. Jahrgang, Heft 4. Bonn: Kirschbaum Verlag, 2002.

SAYDA, FLORIAN: *Zur Aktualisierung von Geodaten eines LBS unter Einbeziehung der Nutzer.* Dissertation. Fakultät für Bauingenieur- und Vermessungswesen, Universität der Bundeswehr München, 2006.

SCHMIDT, ALBRECHT / GELLERSEN, HANS-WERNER: *Modell, Architektur und Plattform für Informationssysteme mit Kontextbezug.* In: Informatik Forschung und Entwicklung, Band 16, Nr. 4. Heidelberg: Springer, 2001.

SCHNEIDER, SABINE / RICCI, FRANCESCO / VENTURINI, ADRIANO / NOT, ELENA: *Usability Guidelines for WAP-based Travel Planning Tools.* In: Information and Communication Technologies in Tourism 2010. Proceedings of the International Conference in Lugano. Wien: Springer Verlag, 2010.

SCHULMEISTER, ROLF: *Grundlagen hypermedialer Lernsysteme: Theorie – Didaktik – Design.* 3. Auflage. München: Oldenbourg Verlag, 2002.

SCHWAB, INGO / POHL, WOLFGANG: *Learning information interest from positive examples.* Proceedings of the seventh international conference on User modeling. New York: Springer, 1999.

SEEBOERGER-WEICHSELBAUM, MICHAEL: *Programmieren mit Eclipse 3. Universelle Entwicklungsplattform für Java, PHP, C/C++, SQL, XML, XSLT, XSL-FO, JSP, Servlets, J2EE, UML, HTML, CSS, JavaScript*. Heidelberg: Mitp-Verlag, 2006.

SHANNON, CLAUDE E. / WEAVER, WARREN: *Mathematical Theory of Communication.* Urbana: University of Illinois Press Sickendiek, 1949.

SJURTS, INSA: *Gabler Lexikon Medienwirtschaft*. 2. Auflage. Wiesbaden: Gabler Verlag, 2011.

STÖCKL, RALPH / GRAU, CHRISTOPH / HESS, THOMAS: *User Generated Content*. In: MedienWirtschaft, Jahrgang 3, Heft 04. Hamburg: New Business Verlag, 2006.

TITKOV, LEONID / POSLAD, STEFAN: *Supporting privacy for U-Commerce tourism services*. UbiAgents Workshop auf der AAMAS-2004 Konferenz. New York: Juli 2004.

VAN WELIE, M. / DE RIDDER, G.: Designing *for Mobile Devices: a Context- Oriented Approach.* IBC Conference "Usability for Mobile Devices". London: Mai 2001.

WOLODTSCHENKO, ALEXANDER: *Atlaskartosemiotik: neue Möglichkeiten und Lösungen*. In: meta - carto-semiotics, Journal for Theoretical Cartography, Vol. 3, 2010.

WORMS, NIKOLAI: *Zur Ökonomie des User-Generated Content*. Bachelorarbeit. Hochschule Furtwangen, 2008.

ZIPF, ALEXANDER: *Forschungsfragen zur benutzer- und kontextangepassten Kartengenerierung für mobile Systeme.* In: Kartographische Nachrichten, 53. Jahrgang, Heft 1. Bonn: Kirschbaum Verlag, 2003.

ZIPF, ALEXANDER: *Nutzungspotenziale und Herausforderungen von „Volunteered Geography". Zur Kombination von GDI-Technologie und nutzergenerierten Geomassendaten*. In: Tagungsband zum Deutschen Geographentag 2009 in Wien. Kiel: Deutsche Gesellschaft für Geographie, 2009.

ZIPF, ALEXANDER: *User-Adaptive Maps for Location-Based Services (LBS) for Tourism*. In: Information and communication technologies in tourism 2002. Proceedings of the International Conference. Innsbruck: Springer Verlag, 2002.

Internetquellen

ANDROIDPIT: *Android Apps, Tests, News Blog und Forum.*
URL: http://www.androidpit.de [letzter Zugriff: 22.01.2011].

ANDROID DEVELOPERS. URL: http://developer.android.com/index.html [letzter Zugriff: 02.04.2011].

DISTIMO: *Comparisons and Contrasts: Windows Phone 7 Marketplace and Google Android Market.* 2011. URL: http://www.distimo.com/publications [letzter Zugriff: 31.03.2011].

EDITION SÄCHSISCHE ZEITUNG: *Wanderkalender 2011 - Schlösser um Dresden.*
URL: http://www.editionsz.de/catalog/product_info.php?products_id=572 [letzter Zugriff: 05.01.2011].

EVERYTRAIL: *GPS Reise-Community***.** URL: http://de.everytrail.com [letzter Zugriff: 29.12.2011].

GARTNER NEWSROOM: *Gartner Says Worldwide Mobile Device Sales to End Users Reached 1.6 Billion Units in 2010. Smartphone Sales Grew 72 Percent in 2010*. 2011.
URL: http://www.gartner.com/it/page.jsp?id=1543014 [letzter Zugriff: 29.03.2011].

GLOBALMOTION. 2010. URL: http://www.globalmotion.com/ [letzter Zugriff: 03.01.2011].

GPS BUSINESS NEWS: *Joost Schreve, everytrail.com: "We are on a path to derive significant ad revenues".* 2009. URL: http://gpsbusinessnews.com/Joost-Schreve,-everytrail-com-We-are-on-a-path-to-derive-significant-ad-revenues_a1615.html [letzter Zugriff: 03.01.2011].

ITUNES: *Everything you need to be entertained*.
URL: http://www.apple.com/itunes/ [letzter Zugriff: 22.01.2011].

KML TUTORIAL: *KML - Google Code*. 2011.
URL: http://code.google.com/intl/de-DE/apis/kml/documentation/kml_tut.html [letzter Zugriff: 02.04.2011].

MOBILE ATLAS CREATOR: *Prepare online maps for your mobile device*. 2008.
URL: http://mobac.dnsalias.org/index.html [letzter Zugriff: 27.03.2011].

MOBILE MARKETING ASSOCIATION: *Mobile Applications.* 2008.
URL: http://www.mmaglobal.com/mobileapplications.pdf [letzter Zugriff: 03.01.2011].

OSMDROID: *OpenStreetMap-Tools for Android*.
URL: http://code.google.com/p/osmdroid/ [letzter Zugriff: 14.03.2011].

SOURCEFORGE: *Downloads suspended - for now*. 2011.
URL: http://sourceforge.net/projects/mobac/forums/forum/861096/topic/4423122 [letzter Zugriff: 27.03.2011].

WIKIPEDIA: Die freie Enzyklopädie.
URL: http://de.wikipedia.org/wiki/Wikipedia:Hauptseite [letzter Zugriff: 04.04.2011].

WIKITUDE: *Das Handy als Reiseführer – Mobile Applikationen im Tourismus*. 2010.
URL: http://www.wikitude.org/de/dedas-handy-als-reisefhrer-mobile-applikationen-im-tourismus [letzter Zugriff: 02.01.2011].

Quellen der Wanderrouten der entwickelten Applikation

EVERYTRAIL: *Teichstein, Richterschlüchte und Hinteres Raubschloß - Saxony, Germany*.
URL: http://de.everytrail.com/view_trip.php?trip_id=631939 [letzter Zugriff: 18.03.2011].

EVERYTRAIL: *Von Königstein zum Gohrisch und Papststein und zurück - Saxony, Germany*.
URL: http://de.everytrail.com/view_trip.php?trip_id=631968 [letzter Zugriff: 18.03.2011].

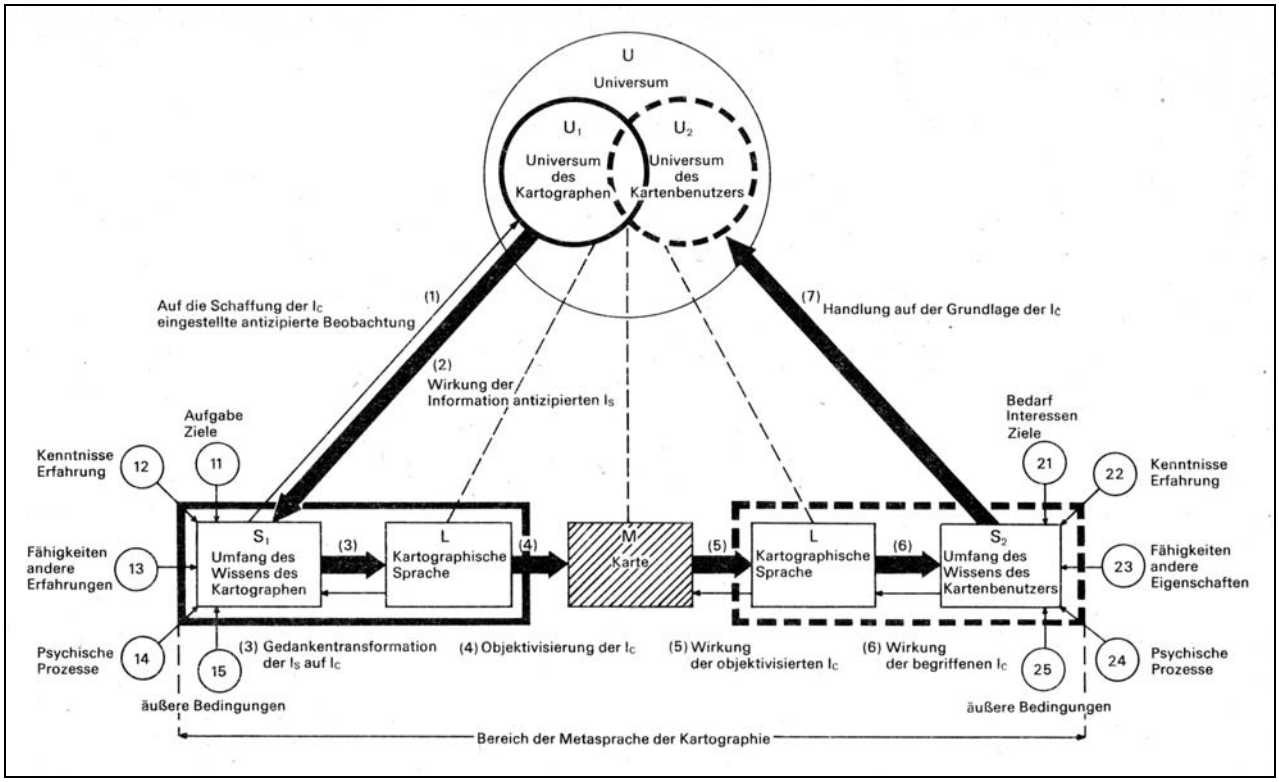
EVERYTRAIL: *Wandern Schrammsteine - Sachsen, Germany.*
URL: http://de.everytrail.com/view_trip.php?trip_id=62778 [letzter Zugriff: 18.03.2011].

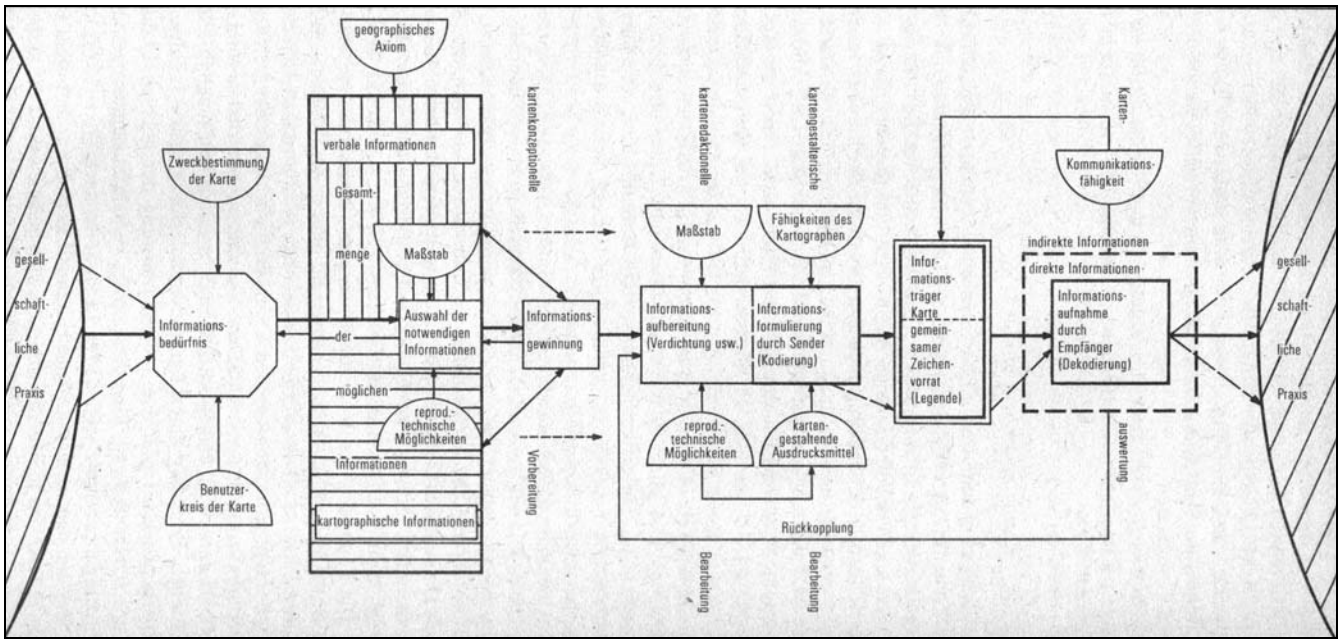
INSTITUT FÜR KARTOGRAPHIE: *Wanderkalender 2010 - Wandern östlich der Elbe*.
URL: http://kartographie.geo.tu-dresden.de/extern/kalender2010/index.php?l=ger [letzter Zugriff: 28.02.2011]

WANDERKALENDER 2010: *Wanderkalender 2010. Wandern östlich der Elbe*. Dresden: edition Sächsische Zeitung, 2009.

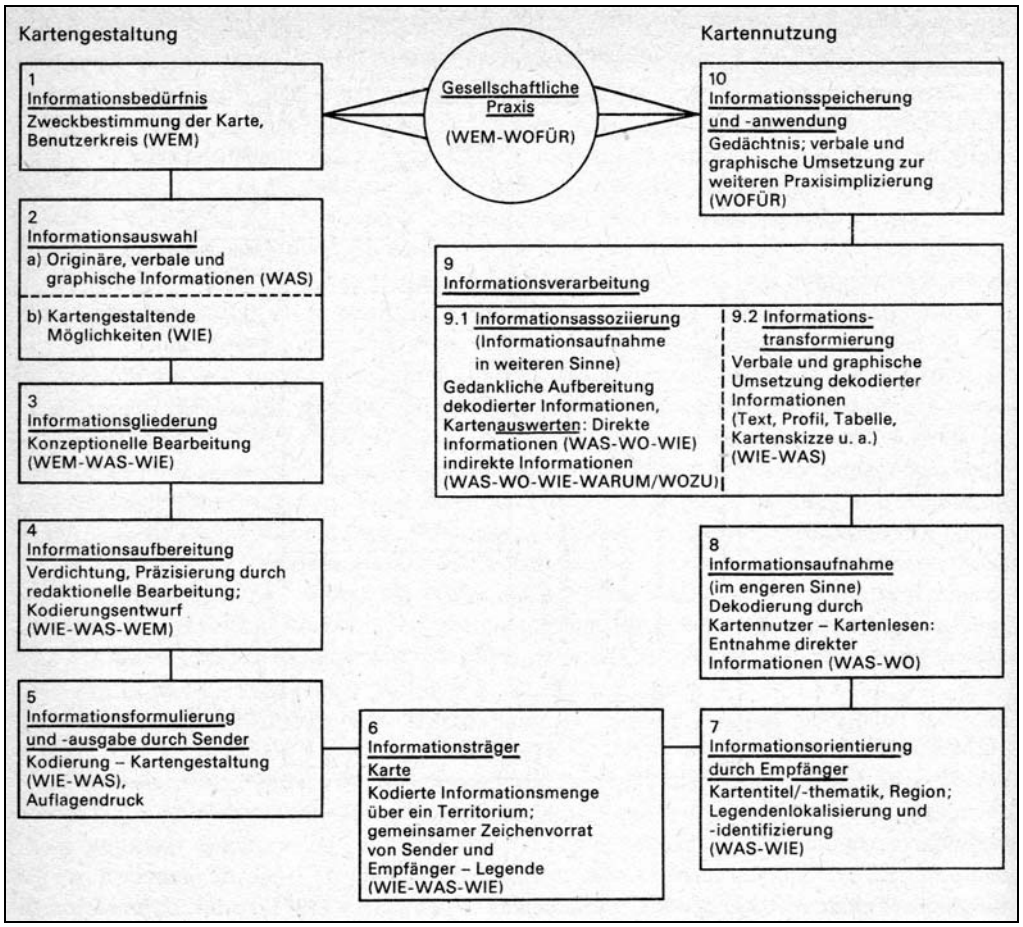
Anhang

Anhangsverzeichnis


U
Universum
U1
Universum des Kartographen
U2
Universum des Kartenbenutzers
(1)
Auf die Schaffung der Ic eingestellte antizipierte Beobachtung
(2)
Wirkung der Information antizipierten Is
(7)
Handlung auf der Grundlage der Ic
Aufgabe
Ziele
11
Kenntnisse
Erfahrung
12
Fähigkeiten
andere
Erfahrungen
13
Psychische
Prozesse
14
15
S1
Umfang des Wissens des Kartographen
(3)
L
Kartographische Sprache
(4)
M
Karte
(5)
L
Kartographische Sprache
(6)
S2
Umfang des Wissens des Kartenbenutzers
Bedarf
Interessen
Ziele
21
Kenntnisse
Erfahrung
22
Fähigkeiten
andere
Eigenschaften
23
Psychische
Prozesse
24
25
(3) Gedankentransformation der Is auf Ic
(4) Objektivisierung der Ic
(5) Wirkung der objektivisierten Ic
(6) Wirkung der begriffenen Ic
äußere Bedingungen
äußere Bedingungen
Bereich der Metasprache der Kartographie



(OGRISSEK, 1987)



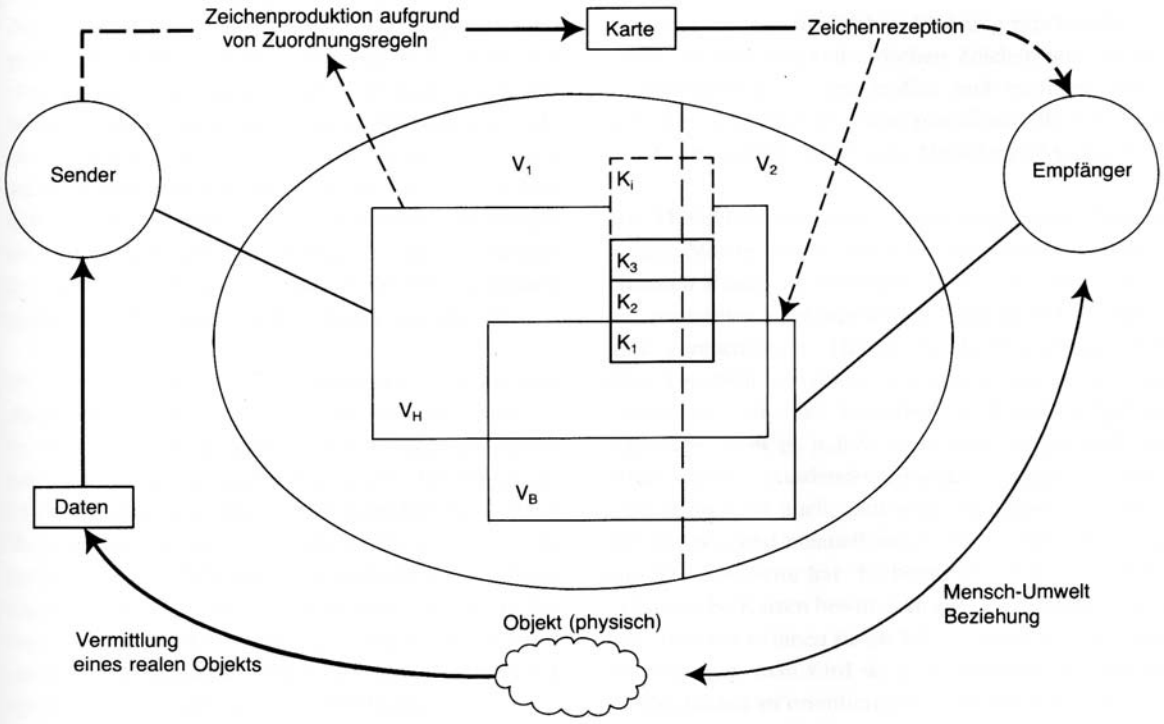
(OGRISSEK, 1987)

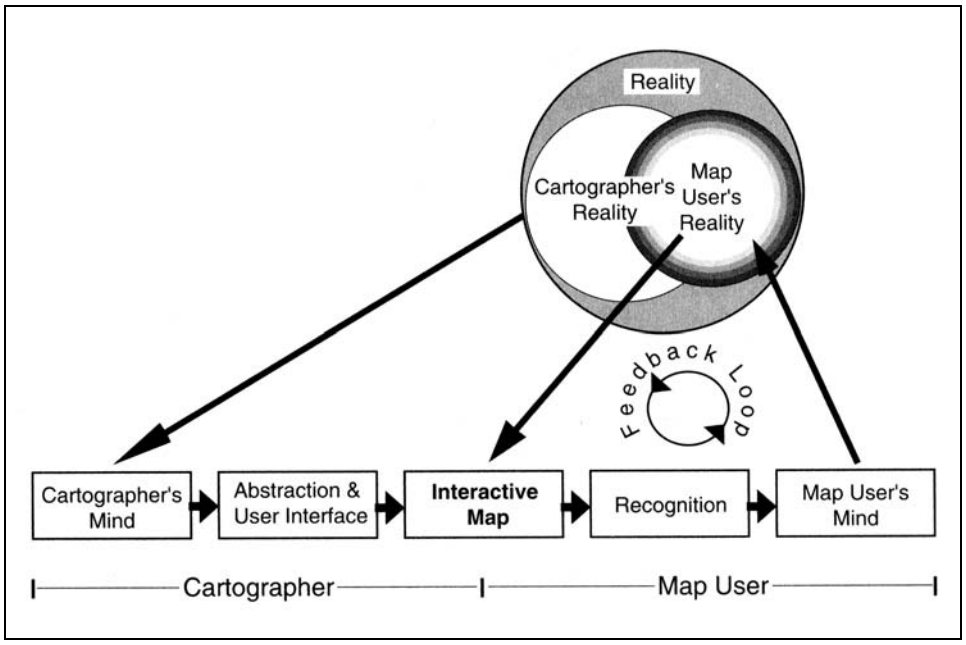
I. Zeichenprozeßschema

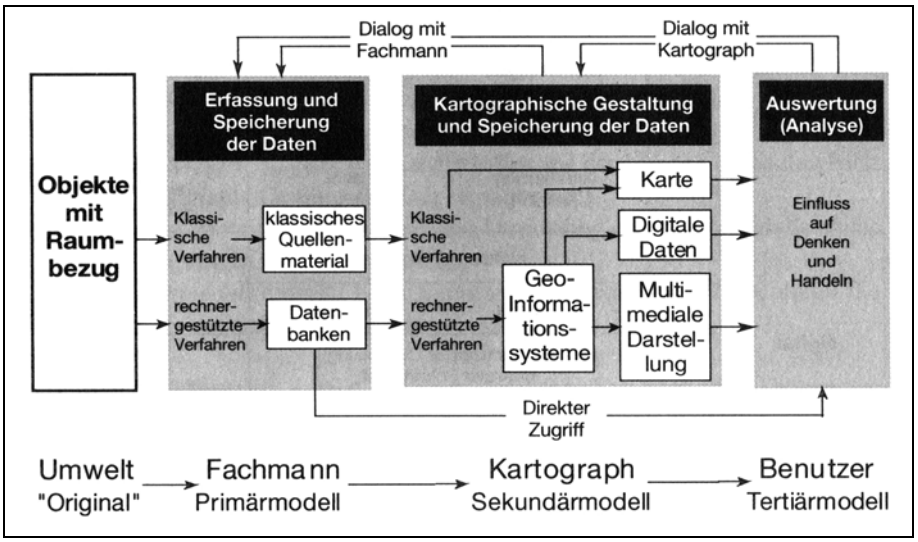
V Gesamtheit möglicher raumbezogener Vorstellungen
- V_1 fehlerfreier Bereich
- V_2 fehlerbehafteter Bereich
- V_H Bewußtseinsinhalte des Kartenherstellers
- V_B Bewußtseinsinhalte des Kartenbenutzers

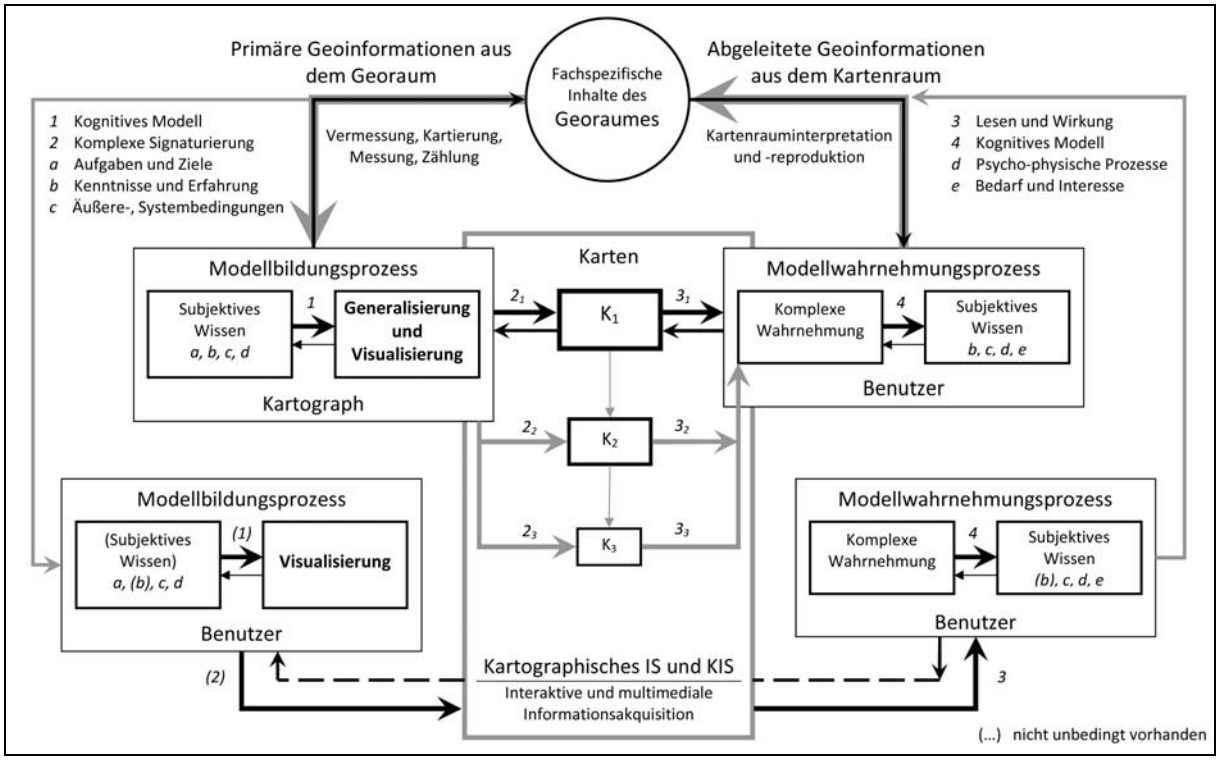
K in die kartographische Darstellung eingegangene Bewußtseinsinhalte
- K_1 redundante Bestandteile für den Benutzer, aber Grundlage für jeden Informationsgewinn
- K_2 direkter Informationsgewinn für den Benutzer
- K_3 vom Benutzer nicht verstandene Informationen
- K_i indirekter Informationsgewinn für den Benutzer bzw. teilweise auch für den Hersteller

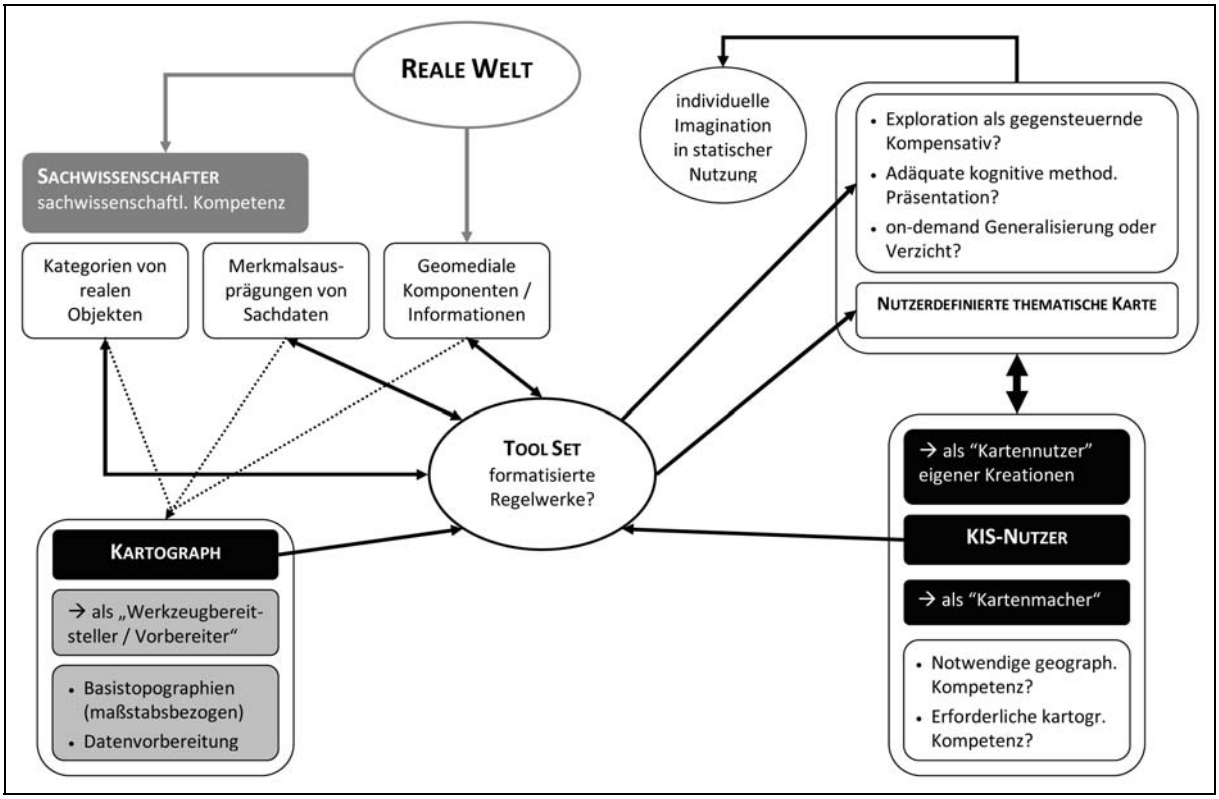
II. Übertragungsschema →




Reality
Cartographer's Reality
Map User's Reality
Feedback Loop
Cartographer's Mind
Abstraction & User Interface
Interactive Map
Recognition
Map User's Mind
Cartographer
Map User

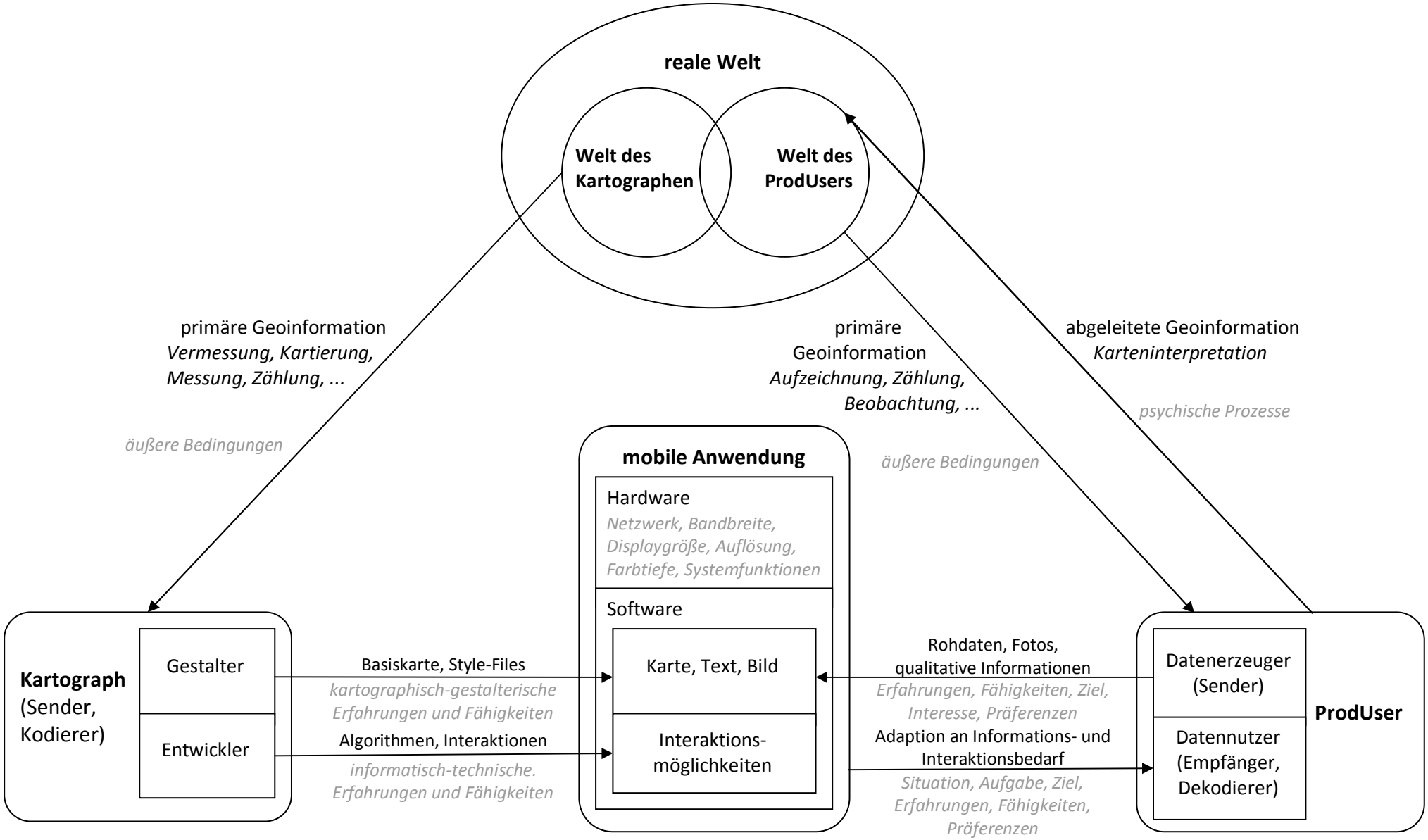

Dialog mit Fachmann
Dialog mit Kartograph
Objekte mit Raum-bezug
Erfassung und Speicherung der Daten
Kartographische Gestaltung und Speicherung der Daten
Auswertung (Analyse)
Klassi-sche Verfahren
klassisches Quellen-material
rechner-gestützte Verfahren
Daten-banken
Klassi-sche Verfahren
rechner-gestützte Verfahren
Geo-Informa-tions-systeme
Karte
Digitale Daten
Multi-mediale Darstel-lung
Einfluss auf Denken und Handeln
Direkter Zugriff
Umwelt "Original"
Fachmann Primärmodell
Kartograph Sekundärmodell
Benutzer Tertiärmodell


Primäre Geoinformationen aus dem Georaum
Fachspezifische Inhalte des Georaumes
Abgeleitete Geoinformationen aus dem Kartenraum
1 Kognitives Modell
2 Komplexe Signaturierung
a Aufgaben und Ziele
b Kenntnisse und Erfahrung
c Äußere-, Systembedingungen
Vermessung, Kartierung, Messung, Zählung
Kartenrauminterpretation und -reproduktion
3 Lesen und Wirkung
4 Kognitives Modell
d Psycho-physische Prozesse
e Bedarf und Interesse
Modellbildungsprozess
Subjektives Wissen a, b, c, d
1
Generalisierung und Visualisierung
Kartograph
Karten
2₁
K1
3₁
Modellwahrnehmungsprozess
Komplexe Wahrnehmung
4
Subjektives Wissen b, c, d, e
Benutzer
2₂
K2
3₂
2₃
K3
3₃
Modellbildungsprozess
(Subjektives Wissen) a, (b), c, d
(1)
Visualisierung
Benutzer
Modellwahrnehmungsprozess
Komplexe Wahrnehmung
4
Subjektives Wissen (b), c, d, e
Benutzer
(2)
Kartographisches IS und KIS
Interaktive und multimediale Informationsakquisition
3
(...) nicht unbedingt vorhanden



(nach KELNHOFER, 2003)

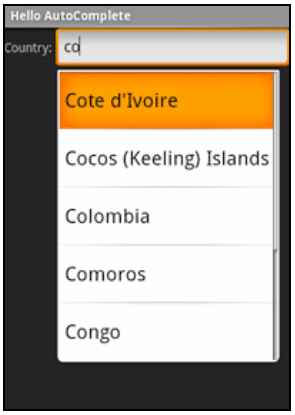
Autor	Jahr	Titel	Art des Modells	Art der Kommunikation
Shannon/Weaver	1949	„Allgemeines Kommunikationsmodell“	Systemmodell	Simplex-Kommunikation
Koláčný	1969	„Prozess der Kommunikation der kartographischen Information“	Systemmodell	Simplex-Kommunikation
Ogrissek	1974	„Die determinierenden Faktoren der kartographischen Kommunikationskette“	Systemmodell (Ansätze eines Prozessmodells)	Simplex-Kommunikation
Kugler	1976	„Die Karte im Kommunikationsprozess“	Prozessmodell	Simplex-Kommunikation
Breetz	1982	„Kartographische Kommunikationskette“	Prozessmodell	Simplex-Kommunikation
Prell	1983	„Kartographisches Kommunikationsmodell“	Systemmodell	Simplex-Kommunikation
Peterson	1995	„Modell der kartographischen Kommunikation in Verbindung mit Interaktion“	Systemmodell	Duplex-Kommunikation
Freitag	2000	„Informationsfluss der automatengestützten kartographischen Kommunikation“	Systemmodell	Simplex-Kommunikation
Lechthaler	2000	„Interaktive und multimediale Kommunikation“	Systemmodell	Duplex-Kommunikation
Freitag	2001	„Die kognitiven Prozesse der kartographischen Kommunikation“	Prozessmodell (Ansätze eines Systemmodells)	Simplex-Kommunikation
Hake/Grünreich/Meng	2002	„Das kartographische Kommunikationsnetz“	Prozessmodell (Ansätze eines Systemmodells)	Duplex-Kommunikation
Kelnhofer	2003	„Professionelle Geo-Informationsaufbereitung via thematischer Karten“	Systemmodell	Simplex-Kommunikation
Kelnhofer	2003	„Nutzerbestimmte thematische Karten“	Systemmodell	Duplex-Kommunikation


reale Welt
Welt des Kartographen
Welt des ProdUsers
primäre Geoinformation
Vermessung, Kartierung, Messung, Zählung, ...
äußere Bedingungen
primäre Geoinformation
Aufzeichnung, Zählung, Beobachtung, ...
äußere Bedingungen
abgeleitete Geoinformation
Karteninterpretation
psychische Prozesse
mobile Anwendung
Hardware
Netzwerk, Bandbreite, Displaygröße, Auflösung, Farbtiefe, Systemfunktionen
Software
Karte, Text, Bild
Interaktions-möglichkeiten
Kartograph (Sender, Kodierer)
Gestalter
Entwickler
Basiskarte, Style-Files
kartographisch-gestalterische Erfahrungen und Fähigkeiten
Algorithmen, Interaktionen
informatisch-technische. Erfahrungen und Fähigkeiten
Rohdaten, Fotos, qualitative Informationen
Erfahrungen, Fähigkeiten, Ziel, Interesse, Präferenzen
Adaption an Informations- und Interaktionsbedarf
Situation, Aufgabe, Ziel, Erfahrungen, Fähigkeiten, Präferenzen
Datenerzeuger (Sender)
Datennutzer (Empfänger, Dekodierer)
ProdUser

	outdooractive	***Qype***	***TravelBook Berlin***	***Vienna Travel Guide***
Wird die Position des Benutzers berücksichtigt?	Ja, aktuelle Position des Nutzers in Karte anzeigbar	Ja, aktuelle Position des Nutzers in Karte anzeigbar	Ja, aktuelle Position des Nutzers in Karte anzeigbar Entfernung des Benutzers zu Objekten kann abgefragt werden	Nein
Sind die Karte und andere Inhalte miteinander verknüpft?	Ja, aus Listenansicht ausgewählte Route wird in Karte angezeigt Informationen zu ausgewähltem POI in Karte abrufbar	Ja, aus Listenansicht ausgewählter POI in Karte anzeigbar Informationen zu ausgewähltem POI in Karte abrufbar	Ja, aus Listenansicht ausgewählter POI in Karte anzeigbar Namen von ausgewähltem POI in Karte abrufbar Informationen zu selbstdefinierten Pins in Karte aufrufbar	Ja, aus Listenansicht ausgewählter POI in Karte anzeigbar Informationen zu ausgewähltem POI in Karte abrufbar
Wird eine vom Entwickler selbst erstellte Karte verwendet?	Nein (*Google Maps*)	Nein (*Google Maps*)	Ja	Ja
Besteht die Möglichkeit, das Kartendesign zu ändern?	Ja, Wahl zwischen Satellitenbild und Straßenkarte	Nein	Ja, Wahl zwischen Straßen-, Wander- und Nachtkarte	Nein
Besteht die Möglichkeit, Karteninhalte ein- und auszublenden?	Nein	Nein	Ja, Höhenlinien und *Wikipedia*-Artikel können ein- und ausgeblendet werden	Ja, POIs können kategorienweise ein- und ausgeblendet werden können
Sind neben Zoomen und Verschieben weitere Interaktionen mit der Basiskarte möglich?	Nein	Nein	Ja, Drehen und Norden	Nein
Können die Inhalte durch den Benutzer personalisiert werden?	Nein	Nein	Ja, eigene Pins und Marker können definiert und mit Informationen versehen werden	Nein
Besteht die Möglichkeit der Dateneingabe?	Nein	Ja, neue POIS sowie Beiträge oder Fotos zu bestehenden POIs können eingegeben werden	Nein	Nein
Verfügt die Applikation über eine Suchfunktion?	Ja (Online-Suche)	Ja (Online-Suche)	Ja	Ja
Verfügt die Applikation über eine interne Routing-Funktion?	Nein	Nein	Ja	Nein
Wird mit einer Augmented Reality gearbeitet?	Nein	Nein	Nein	Nein
Kann die Applikation auch ohne Internetverbindung uneingeschränkt verwendet werden?	Nein, Kartenmaterial und sämtliche Touren werden aus dem Internet bezogen	Nein, Kartenmaterial und sämtliche Informationen werden aus dem Internet bezogen	Nein, Routing funktioniert nur online und einige *Wikipedia*-Artikel sind offline nicht verfügbar	Ja

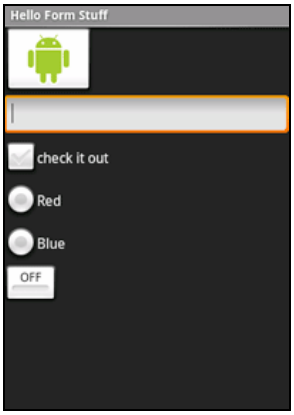
	Allgäu	***Lonely Planet Berlin***	***Lonely Planet Paris***	***Osteseeküste - ADAC Wanderführer***	***The North Face Trailhead***
Wird die Position des Benutzers berücksichtigt?	Ja, aktuelle Position des Nutzers in Karte anzeigbar Entfernung des Benutzers zu POIs kann abgefragt werden	Ja, aktuelle Position des Nutzers in Karte anzeigbar	Ja, aktuelle Position des Nutzers in Karte anzeigbar Entfernung des Benutzers zu POIs kann abgefragt werden	Ja, aktuelle Position des Nutzers in Karte anzeigbar Entfernung des Benutzers zu Wanderrouten kann abgefragt werden	Ja, aktuelle Position des Nutzers in Karte anzeigbar Entfernung des Benutzers zu Wanderrouten kann abgefragt werden
Sind die Karte und andere Inhalte miteinander verknüpft?	Ja, aus Listenansicht ausgewählter POI in Karte anzeigbar Informationen zu ausgewähltem POI in Karte abrufbar	Ja, aus Listenansicht ausgewählter POI in Karte anzeigbar Informationen zu ausgewähltem POI in Karte abrufbar Verlinkungen in Fließtexten zu Karte vorhanden	Ja, aus Listenansicht ausgewählter POI in Karte anzeigbar Informationen zu ausgewähltem POI in Karte abrufbar Verlinkungen in Fließtexten zu Karte vorhanden	Ja, aus Listenansicht ausgewählte Route wird in Karte angezeigt Informationen zu ausgewähltem POI in Karte abrufbar	Informationen zu ausgewähltem Objekt in Karte abrufbar Foto zu ausgewähltem Pin in Karte aufrufbar
Wird eine vom Entwickler selbst erstellte Karte verwendet?	Ja, neben *Google Maps*	Ja	Ja	Ja	Nein (*Google Maps*)
Besteht die Möglichkeit, das Kartendesign zu ändern?	Ja, Wahl zwischen Karte von *Alpstein*, *Google Maps*-Straßenkarte und *Google Maps*-Hybrid	Nein	Nein	Nein	Ja, Wahl zwischen Satellitenbild und Straßenkarte
Besteht die Möglichkeit, Karteninhalte ein- und auszublenden?	Nein	Ja, POIs können kategorienweise ein- und ausgeblendet werden können	Ja, POIs können kategorienweise ein- und ausgeblendet werden können	Nein	Nein
Sind neben Zoomen und Verschieben weitere Interaktionen mit der Basiskarte möglich?	Nein	Nein	Nein	Nein	Nein
Können die Inhalte durch den Benutzer personalisiert werden?	Nein	Nein	Ja, einzelne POIs können zu Favoriten ernannt werden	Nein	Ja, einzelne Routen können zu Favoriten ernannt werden
Besteht die Möglichkeit der Dateneingabe?	Nein	Nein	Nein	Nein	Ja, eigene Touren können erstellt und veröffentlicht werden
Verfügt die Applikation über eine Suchfunktion?	Nein	Nein	Ja	Nein	Ja (Online-Suche)
Verfügt die Applikation über eine interne Routing-Funktion?	Nein, nur Verweis auf *Google Maps*	Nein	Nein	Nein, nur Verweis auf *Google Maps*	Ja
Wird mit einer Augmented Reality gearbeitet?	Nein	Nein	Nein	Ja, Anzeige umliegender Gipfel	Nein
Kann die Applikation auch ohne Internetverbindung uneingeschränkt verwendet werden?	Nein, Kartenmaterial und Informationen zu Wetter oder aktuellen Veranstaltungen im Allgäu werden aus dem Internet bezogen	Ja	Ja	Nein, da die Routing-Funktion von *Google Maps* genutzt wird	Nein, Kartenmaterial und sämtliche Touren werden aus dem Internet bezogen

AutoComplete

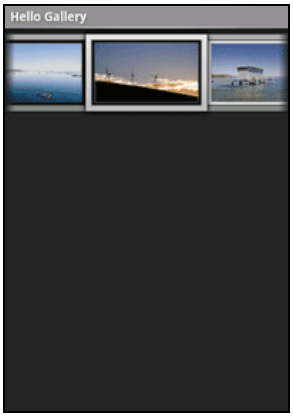


DatePicker

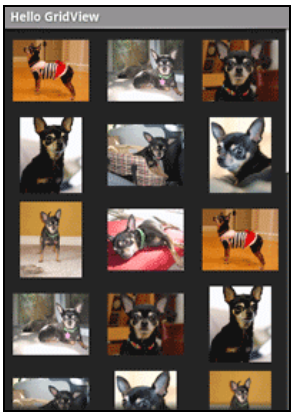
FormStuff



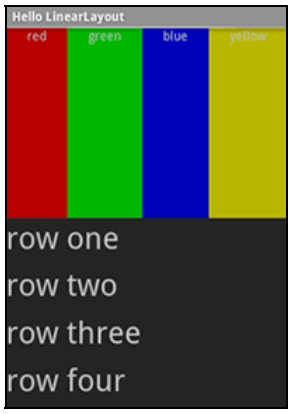
Gallery



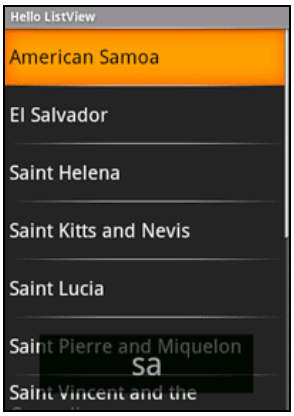
GridView



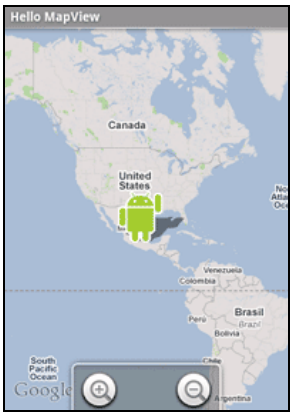
LinearLayout



ListView

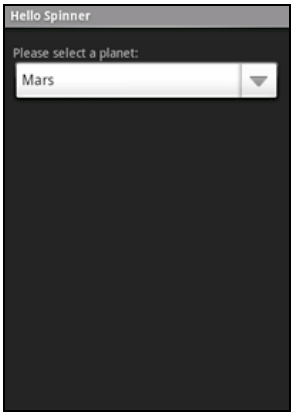


MapView



RelativeLayout

Spinner



TableLayout

TabWigdet

TimePicker

WebView

Steinbruchpfad Wehlen

Die Erschließung des Steinbruchpfades

Bei der im folgenden vorgestellten Wanderroute handelt es sich um den **Steinbruchpfad** der Sächsischen Schweiz, der erst vor reichlich 20 Jahren als Wanderweg erschlossen wurde. Er beginnt am Aufstieg zur Wilkeaussicht bei Stadt Wehlen und endet am Fährhaus Zeichen. Auf einer Strecke von knapp drei Kilometern kann man auf den Spuren der Steinbrecher und bedeutender Künstler wie Elfriede Lohse-Wächtler, Pol Cassel und Otto Dix wandeln und vielen Zeugen der sächsischen Steinbruchgeschichte begegnen.

Wegstein des Steinbruchpfades

Einst war dieser **Haldenweg** eine Verbindung zwischen den zahlreichen Steinbrüchen und ist auch heute noch ein Notweg bei Elbhochwasser. Bereits Anfang der 1980er Jahre wurde durch den Nationalparkführer Andreas Bartsch dieser **Lehrpfad** angelegt. Bis heute bemüht er sich um die ständige Erweiterung und Pflege einer an der Alten Steinsäge zu bewundernden Sammlung von Exponaten zur Steinbruchgeschichte, die Erhaltung der Wegmarkierungen und das Aufstellen neuer Wegsteine. Informationen zu seinen sachkundigen und humorvollen Führungen sowie zum Steinbruchpfad selbst findet man im Internet unter www.steinbruchfuehrungen.de.

Es ist zu beachten, dass der Steinbruchpfad fast vollständig über **Privatland** führt, das Begehen geschieht somit auf eigene Gefahr. Der markierte Weg sollte nicht verlassen und der Privatbereich der Anlieger respektiert werden.

Desweiteren ist **Trittsicherheit** erforderlich und bei Schnee und Eis ist vom Begehen des Steinbruchpfades abzuraten.

Die Geschichte der Steinbrecherei in der Sächsischen Schweiz

Die Anfänge der **Sandsteingewinnung** in der Sächsischen Schweiz sind bis heute nicht genau zu datieren. Erste Hinweise liefert jedoch der slawische Wortstamm verschiedener Orts- und Flurnamen aus der Zeit der ersten slawischen Besiedlung, also wurde bereits vor über 1000 Jahren in der Sächsischen Schweiz Sandstein abgebaut. Hirsestampftöpfe aus dieser Zeit sind im **Heimatmuseum** in **Bad Schandau** zu sehen.

Ein größerer und anhaltender Bedarf an Sandstein entwickelte sich mit Beginn der Errichtung von steinernen Bauten wie Kirchen und Burgen. Somit existierte ab dem 12. und 13. Jahrhundert eine ständige Sandsteingewinnung. Im 16. Jahrhundert gab es in der gesamten Sächsischen Schweiz Abbaugebiete.

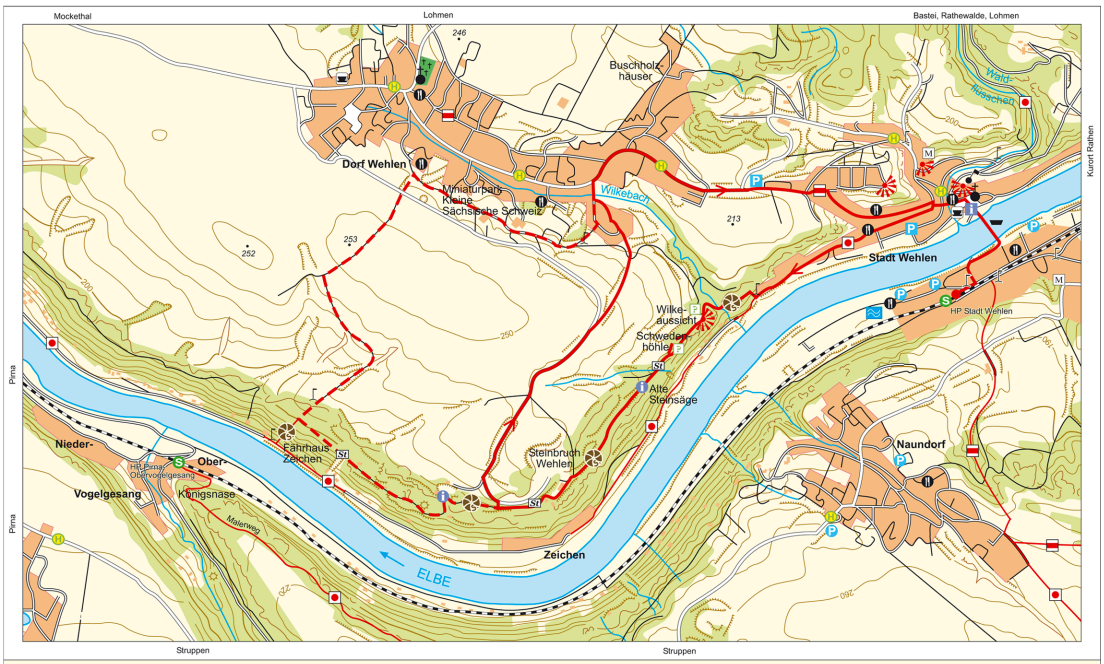
Der Pirnaische Sandstein war weit bekannt und als Baumaterial und für die Herstellung von Mühlsteinen begehrt. Die Steinbrecherei war zur Existenzgrundlage vieler Menschen geworden. In der Mitte des 16. Jahrhunderts hatten sich die Steinbrüche wegen des steigenden Bedarfs bis zur böhmischen Grenze ausgebreitet.

Die Nachfrage nach Sandstein wurde so hoch, dass es sogar Schwierigkeiten bei der Lieferung gab. Die Steine wurden für die Bauten der Landesfürsten, vor allem in der Residenzstadt Dresden, und für Bildhauerarbeiten benötigt.

Im 18. Jahrhundert ging die Zahl der verfügbaren Steinbrecher drastisch zurück. Gründe dafür waren die Verschlechterung der sozialen Situation der Steinbrecher und die Auswirkungen des Siebenjährigen Krieges. Da aber Arbeitskräfte benötigt wurden, um den anhaltenden Bedarf zu decken, wurden dauerhaft tätige Steinbrecher vom Militärdienst und von Amtsfrondiensten befreit.

Im 19. Jahrhundert wurde das Steinbrecherwesen zunehmend staatlich reglementiert. Um 1900 erlebte die Sandsteingewinnung ihre Blütezeit, danach war ein kontinuierlicher Rückgang zu verzeichnen. Bis heute wird in der Sächsischen Schweiz Sandstein abgebaut, zum Beispiel wurde der Sandstein zum Wiederaufbau der **Dresdner Frauenkirche** im Steinbruch Wehlen gebrochen.

Steinbruch Wehlen



Entlang des Steinbruchpfads Wehlen

- empfohlene Wanderroute mit Startpunkt und Richtungsangabe
- alternative Wanderroute
- markierter Wanderweg
- Steinbruchpfad
- Hauptstraße
- Nebenstraße
- Weg
- Eisenbahn mit S-Bahn-Haltepunkt
- Bushaltestelle, Fähre
- großer Parkplatz, kleiner Parkplatz
- Informationsbüro, Informationstafel
- Informationsschild, Wegweiser
- Museum
- Aussichtspunkt
- Kirche, Burgruine
- Gaststätte, Café
- Freibad
- Höhenlinie mit Höhenangabe in m
- Höhenpunkt in m
- Felsen, Schlucht
- Böschung
- Steinbruchanlage

0 200 400 600 800 1000 m

Maßstab 1 : 20 000

Kartenherstellung:
TU Dresden, Institut für Kartographie, Eva Hauthal

Grundlage:
TK 25 mit Wanderwegen, mit Erlaubnis des Staatsbetriebes Geobasisinformation und Vermessung Sachsen 2009, Erlaubnisnr.: 12088/2009

Entlang des Steinbruchpfads Wehlen

Wegstrecke: Stadt Wehlen – Steinbruchpfad – Dorf Wehlen – Stadt Wehlen
Anfahrt: S-Bahn S1 bis Haltepunkt Stadt Wehlen
Weglänge: 7 km
Dauer: 2,5 Stunden
Höhenunterschied: 150 m

Von Stadt Wehlen zur Wilkeaussicht *[30 min]*

Die empfohlene Wanderung beginnt und endet am Haltepunkt **Stadt Wehlen**. Von dort aus führt ein Weg in Richtung Elbfähre, mit welcher man auf die andere Flussseite gelangt. Nun überquert man den **Marktplatz** der Stadt Wehlen. Dort befindet sich eine der wenigen **Radfahrerkirchen** Deutschlands.

Wilkeaussicht

Die Wanderroute verläuft weiter auf der Pirnaer Straße, die der Elbe stromabwärts folgt, aus dem Zentrum der Stadt Wehlen hinaus und vorbei am städtischen Friedhof. Nach etwa einem Kilometer geht ein Pfad zur **Wilkeaussicht** von der Straße rechts ab, hier beginnt der Steinbruchpfad. Dieser ist markiert mit dem Logo eines Schleifsteines, welches dem Wanderer an Bäumen, auf Felsen und Wegsteinen stets zuverlässig den Weg weist. Nach einem kurzen Anstieg erreicht man die **Wilkeaussicht**, die einen Blick nach Stadt Wehlen, den Weißen Brüchen, dem Basteigebiet und dem Rauenstein erlaubt.

Von der Wilkeaussicht nach Dorf Wehlen *[70 min]*

Von nun an schlängelt sich der Steinbruchpfad an der Kante einer Sandsteinstufe entlang und streift zunächst die **Schwedenhöhle**. Seitlich des Pfades gibt es Einsturzlöcher, die eine Absturzgefahr darstellen, deshalb unbedingt den Pfad nicht verlassen.

Schon bald begegnet man den ersten Zeugen der sächsischen Steinbruchgeschichte, einem alten **Seilrollensockel** eines Bremsberges und der **Alten Steinsäge.** Vor diesem Haus erinnert eine Tafel an die Künstlerin Elfriede Lohse-Wächtler, welche für kurze Zeit hier lebte. An der Alten Steinsäge befindet sich eine Sammlung zahlreicher Exponate aus der Steinbruchgeschichte. Danach führt der Pfad am **Steinbruch Wehlen** vorbei, das Betreten dieses Betriebsgeländes ist allerdings verboten. Bald erreicht man die **Friedrichsbrücke**, einst ein Schüttgerüst für das Abkippen des Abraumes in das Elbtal.

Kurz hinter der Friedrichsbrücke führt rechts eine Treppe am Wegstein 37 vorbei auf die Vorwerkstraße. Der Vorwerkstraße folgt man bis Dorf Wehlen.

Der zweite Abschnitt des Steinbruchpfades kann aus rechtlichen Gründen nur im Rahmen einer Führung

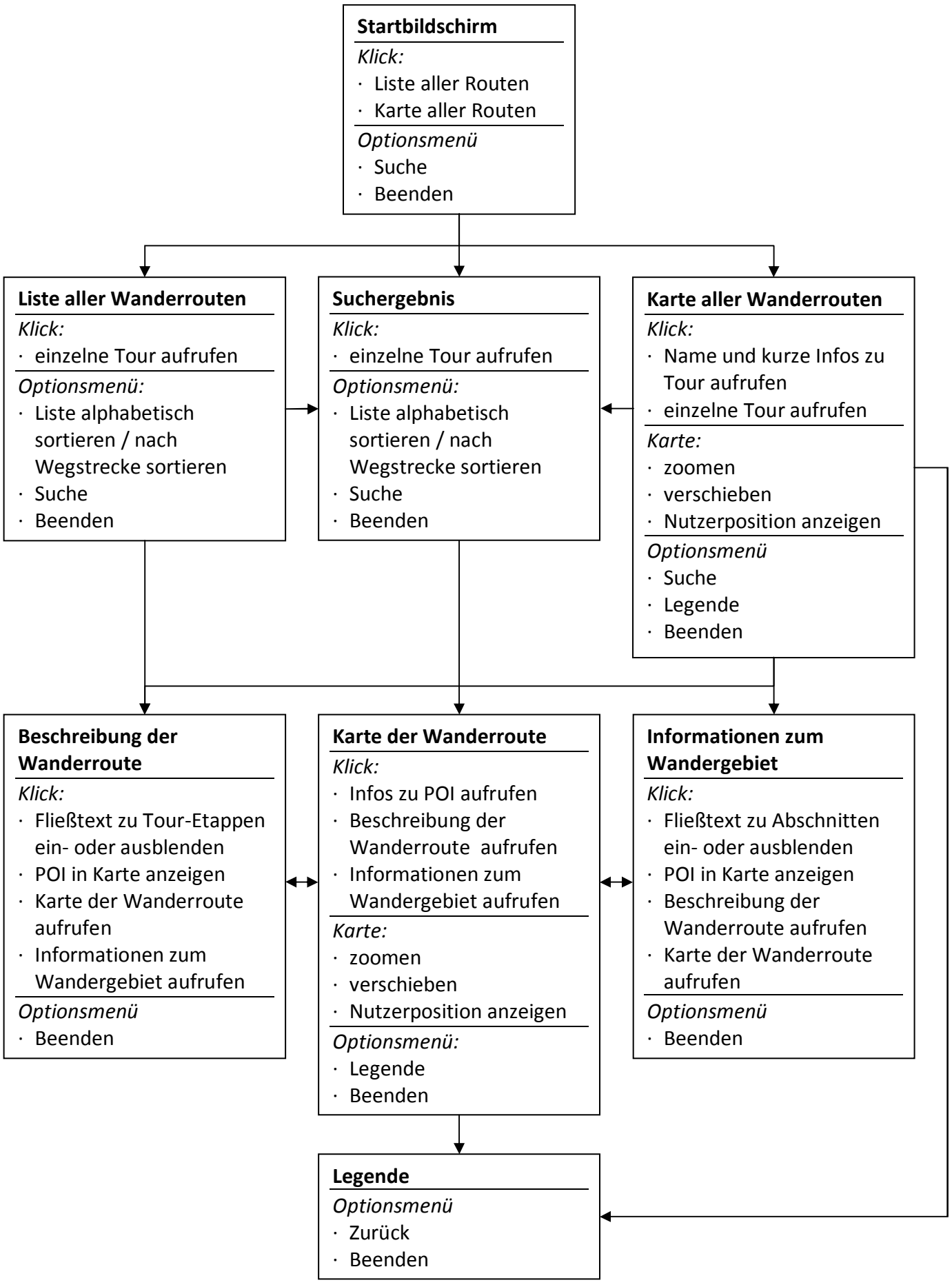
Friedrichsbrücke

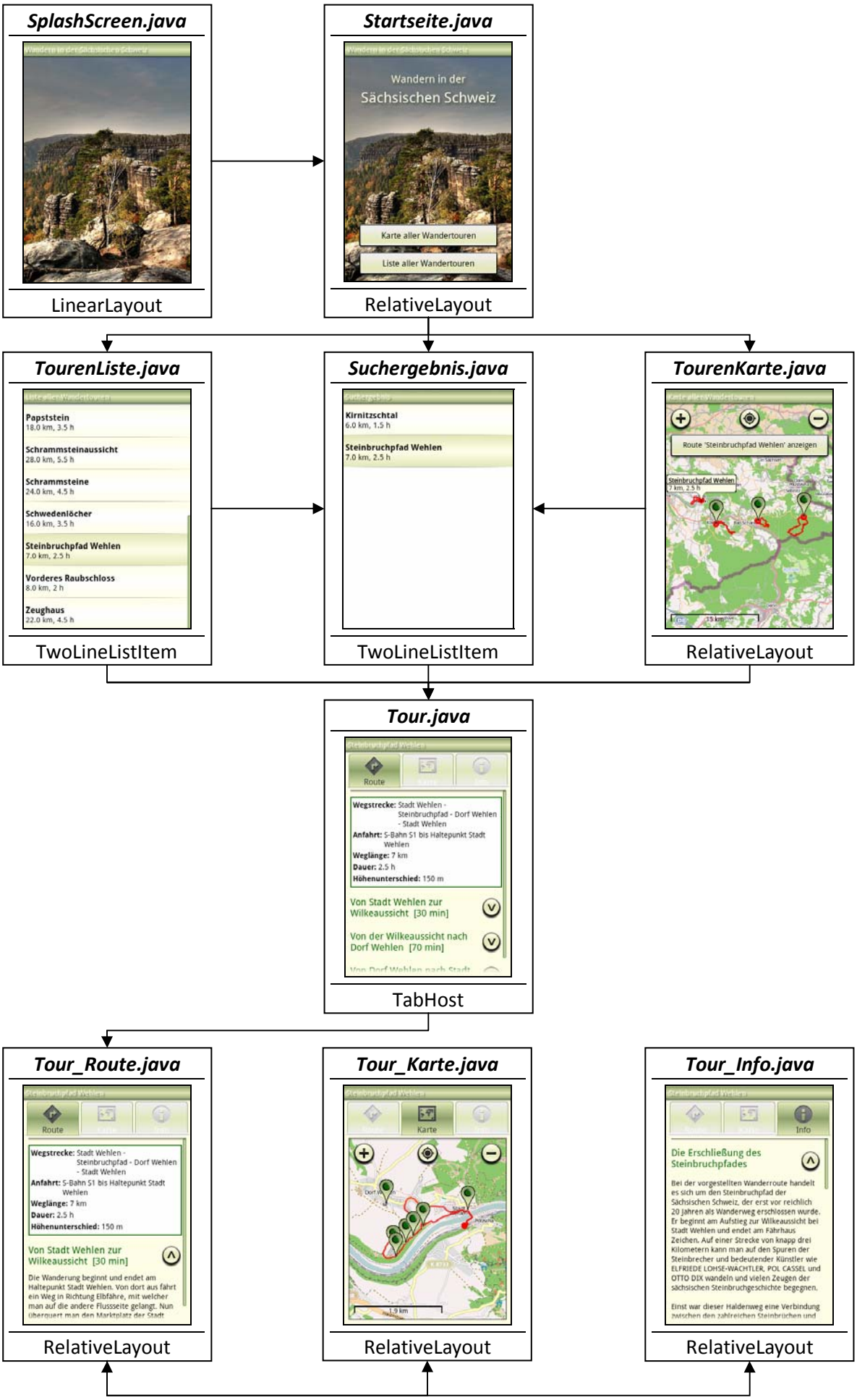
begangen werden. Die in der Karte alternativ eingezeichnete Wanderroute folgt diesem Abschnitt. Er führt an einem ehemaligen **Steinbrecherhaus**, welches später das Atelier des Malers Pol Cassel (1892-1945) war, vorbei sowie an einem Bremsberg, Schleifsteinen, einer **ehemaligen Schmiede** und vielen behauenen Werkstücken.

Von Dorf Wehlen nach Stadt Wehlen *[50 min]*

In Dorf Wehlen verläuft die Wanderroute in der Nähe des Miniaturparks **Kleine Sächsische Schweiz**. Dieser gibt einen Überblick über die Sehenswürdigkeiten und Ausflugsziele der Sächsisch-Böhmischen Schweiz, ist naturgetreu nachgebildet und aus einheimischem Sandstein gefertigt. Bald geht Dorf Wehlen nahtlos in Stadt Wehlen über, wo der Hausbergweg hinunter auf den **Marktplatz** führt. Nun muss man nur noch mit der Fähre die Elbe überqueren, um wieder am Ausgangspunkt der Rundwanderung anzukommen.

Erkundet man den Steinbruchpfad im Rahmen einer **Führung** von Andreas Bartsch, wird diese Wanderung zu einem unvergesslichen Erlebnis.


Startbildschirm
Klick:
· Liste aller Routen
· Karte aller Routen
Optionsmenü
· Suche
· Beenden
Liste aller Wanderrouten
Klick:
· einzelne Tour aufrufen
Optionsmenü:
· Liste alphabetisch sortieren / nach Wegstrecke sortieren
· Suche
· Beenden
Suchergebnis
Klick:
· einzelne Tour aufrufen
Optionsmenü:
· Liste alphabetisch sortieren / nach Wegstrecke sortieren
· Suche
· Beenden
Karte aller Wanderrouten
Klick:
· Name und kurze Infos zu Tour aufrufen
· einzelne Tour aufrufen
Karte:
· zoomen
· verschieben
· Nutzerposition anzeigen
Optionsmenü
· Suche
· Legende
· Beenden
Beschreibung der Wanderroute
Klick:
· Fließtext zu Tour-Etappen ein- oder ausblenden
· POI in Karte anzeigen
· Karte der Wanderroute aufrufen
· Informationen zum Wandergebiet aufrufen
Optionsmenü
· Beenden
Karte der Wanderroute
Klick:
· Infos zu POI aufrufen
· Beschreibung der Wanderroute aufrufen
· Informationen zum Wandergebiet aufrufen
Karte:
· zoomen
· verschieben
· Nutzerposition anzeigen
Optionsmenü:
· Legende
· Beenden
Informationen zum Wandergebiet
Klick:
· Fließtext zu Abschnitten ein- oder ausblenden
· POI in Karte anzeigen
· Beschreibung der Wanderroute aufrufen
· Karte der Wanderroute aufrufen
Optionsmenü
· Beenden
Legende
Optionsmenü
· Zurück
· Beenden


SplashScreen.java
LinearLayout
Startseite.java
Wandern in der Sächsischen Schweiz
Karte aller Wandertouren
Liste aller Wandertouren
RelativeLayout
TourenListe.java
Papststein
18.0 km, 3.5 h
Schrammsteinaussicht
28.0 km, 5.5 h
Schrammsteine
24.0 km, 4.5 h
Schwedenlöcher
16.0 km, 3.5 h
Steinbruchpfad Wehlen
7.0 km, 2.5 h
Vorderes Raubschloss
8.0 km, 2 h
Zeughaus
22.0 km, 4.5 h
TwoLineListItem
Suchergebnis.java
Kirnitzschtal
6.0 km, 1.5 h
Steinbruchpfad Wehlen
7.0 km, 2.5 h
TwoLineListItem
TourenKarte.java
Route 'Steinbruchpfad Wehlen' anzeigen
RelativeLayout
Tour.java
Route
Wegstrecke: Stadt Wehlen - Steinbruchpfad - Dorf Wehlen - Stadt Wehlen
Anfahrt: S-Bahn S1 bis Haltepunkt Stadt Wehlen
Weglänge: 7 km
Dauer: 2.5 h
Höhenunterschied: 150 m
Von Stadt Wehlen zur Wilkeaussicht [30 min]
Von der Wilkeaussicht nach Dorf Wehlen [70 min]
TabHost
Tour_Route.java
Route
Wegstrecke: Stadt Wehlen - Steinbruchpfad - Dorf Wehlen - Stadt Wehlen
Anfahrt: S-Bahn S1 bis Haltepunkt Stadt Wehlen
Weglänge: 7 km
Dauer: 2.5 h
Höhenunterschied: 150 m
Von Stadt Wehlen zur Wilkeaussicht [30 min]
RelativeLayout
Tour_Karte.java
Karte
RelativeLayout
Tour_Info.java
Info
Die Erschließung des Steinbruchpfades
RelativeLayout

Touren

- _id (INTEGER PRIMARY KEY)
- Tour_Name (TEXT)
- Weglaenge_km (NUMERIC)
- Dauer_h (NUMERIC)
- Wegstrecke (TEXT)
- Anfahrt (TEXT)
- Hoehenunterschied_m (NUMERIC)
- Etappe_1_Name (TEXT)
- Etappe_1_Dauer_min (NUMERIC)
- Etappe_1_Text (TEXT)
- Etappe_1_Bild (BLOB)
- Etappe_2_Name (TEXT)
- Etappe_2_Dauer_min (NUMERIC)
- Etappe_2_Text (TEXT)
- Etappe_2_Bild (BLOB)
- Etappe_3_Name (TEXT)
- Etappe_3_Dauer_min (NUMERIC)
- Etappe_3_Text (TEXT)
- Etappe_3_Bild (BLOB)
- Etappe_4_Name (TEXT)
- Etappe_4_Dauer_min (NUMERIC)
- Etappe_4_Text (TEXT)
- Etappe_4_Bild (BLOB)
- Info_1_Name (TEXT)
- Info_1_Text (TEXT)
- Info_1_Bild (BLOB)
- Info_2_Name (TEXT)
- Info_2_Text (TEXT)
- Info_2_Bild (BLOB)
- Info_3_Name (TEXT)
- Info_3_Text (TEXT)
- Info_3_Bild (BLOB)
- Info_4_Name (TEXT)
- Info_4_Text (TEXT)
- Info_4_Bild (BLOB)
- KML (BLOB)

```
package org.me.wanderfuehrer;

import android.content.Context;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteException;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class DataBaseHelper extends SQLiteOpenHelper{

    private static String DB_PATH;
    private static String DB_PATH_PREFIX = "/data/data/";
    private static String DB_PATH_SUFFIX = "/databases/";
    private static String DB_NAME = "wanderfuehrer_db.db";

    private SQLiteDatabase myDataBase;

    private final Context myContext;

    // Konstruktor
    public DataBaseHelper(Context context) {

        super(context, DB_NAME, null, 1);
        this.myContext = context;
    }

    // erzeugt leere DB und ersetzt sie mit eigener DB
    public void createDataBase() throws IOException{

         // setze kompletten Verzeichnis-Pfad, in dem DB auf Smartphone
         // abgelegt werden soll, zusammen
        DB_PATH = DB_PATH_PREFIX + myContext.getPackageName() +
        DB_PATH_SUFFIX + DB_NAME;

        boolean dbExist = checkDataBase();
        SQLiteDatabase db_Read = null;
        Log.d("DB", "Does the database exist already?" + dbExist);

      // wenn DB bereits existiert
     if(dbExist){
           // keine Aktionen nötig
     }
     // andernfalls:
     else{

           //erzeuge leere DB
           db_Read= this.getReadableDatabase();
           db_Read.close();


           try {
```

```
            // kopiere eigene DB und überschreibe leere DB damit
            copyDataBase();

        }
        catch (IOException e) {

            throw new Error("Error copying database");

        }
  }

}


private boolean checkDataBase(){

  SQLiteDatabase checkDB = null;

  try{
        // versuche, DB zu öffnen
        checkDB = SQLiteDatabase.openDatabase( DB_PATH, null,
        SQLiteDatabase.NO_LOCALIZED_COLLATORS);

        checkDB.close();

        // DB existiert
        return true;


  }catch(SQLiteException e){

        // DB existiert noch nicht
        return false;
  }

}


private void copyDataBase() throws IOException{

     // öffne eigene DB als InputStream
     InputStream assetsDB = myContext.getAssets().open(DB_NAME);

    File directory = new File(DB_PATH);
    if (directory.exists() == false )
    {
        directory.mkdir();
    }

  // öffne leere DB als OutputStream
  OutputStream dbOut = new FileOutputStream(DB_PATH);

  // übertrage Bytes von InputSteam in OutputStream
  byte[] buffer = new byte[1024];
  int length;
  while ((length = assetsDB.read(buffer))>0){
        dbOut.write(buffer, 0, length);
```

```
      }

      // schließe Streams
      dbOut.flush();
      dbOut.close();
      dbOut.close();

    }

    public void openDataBase() throws SQLException{

        // öffne DB
        myDataBase = SQLiteDatabase.openDatabase(DB_PATH, null,
        SQLiteDatabase.NO_LOCALIZED_COLLATORS);

    }

    public SQLiteDatabase getDataBase() throws SQLException {

        // öffne und übergebe DB
        myDataBase = SQLiteDatabase.openDatabase(DB_PATH, null,
        SQLiteDatabase.NO_LOCALIZED_COLLATORS);

        return myDataBase;
    }


    @Override
    public synchronized void close() {

        // schließe DB
        if(myDataBase != null)
                myDataBase.close();

        super.close();

    }

    @Override
    public void onCreate(SQLiteDatabase db) {

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {

    }

}
```

```
package org.me.wanderfuehrer;

import android.app.ListActivity;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.os.Handler;
import android.util.AttributeSet;
import android.util.Log;
import android.view.InflateException;
import android.view.LayoutInflater;
import android.view.LayoutInflater.Factory;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ListView;

public class TourenListe extends ListActivity {

      private static final String dbTable = "Touren";
      protected SQLiteDatabase db;
      private DataBaseHelper dbHelper = new DataBaseHelper(this);

      public static String sel_tour;
      public static String searchKeywords;
      boolean sortByAlphabet = true;


    @Override
      public void onCreate(Bundle icicle) {
             super.onCreate(icicle);
             setTitle(R.string.Liste_Touren);

             // ändere Hintergrundfarbe des Optionsmenüs
             setMenuBackground();

             CreateList();

    }

    protected void CreateList() {

        final Cursor dbCursor;

        try {

            // öffne DB
            db = dbHelper.getDataBase();

            // wenn Variable sortByAlphabet wahr ist:
            // rufe Funktion ListOrderedByName() auf
            if (sortByAlphabet == true)
            {
                dbCursor = ListOrderedByName();
            }
```

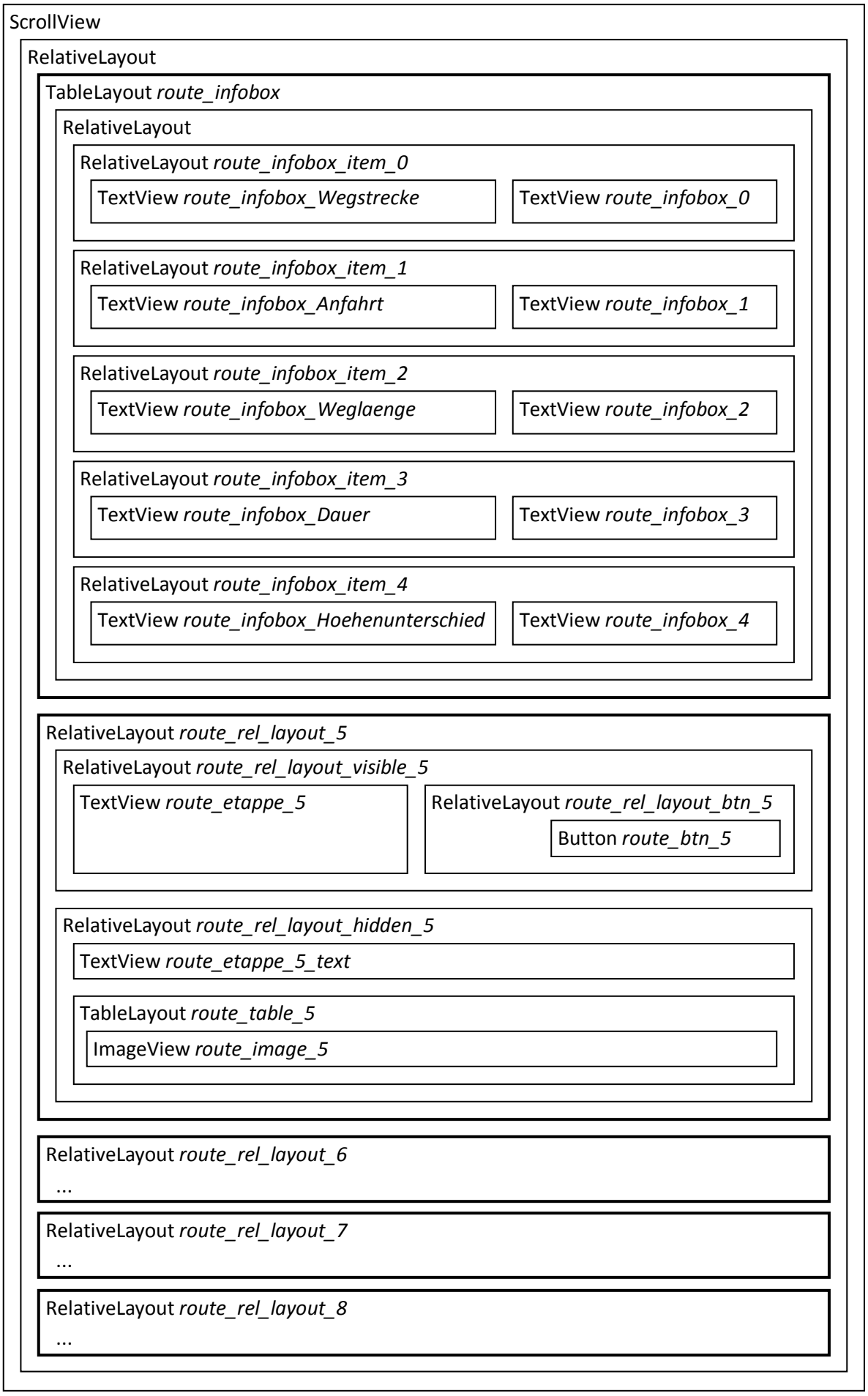
```
        // andernfalls rufe Funktion ListOrderedByDistance() auf
        else
        {
            dbCursor = ListOrderedByDistance();
        }

        int NameColumn  = dbCursor.getColumnIndex("Tour_Name");
        int WeglaengeColumn = dbCursor.getColumnIndex("Weglaenge_km");
        int DauerColumn = dbCursor.getColumnIndex("Dauer_h");

        // Anzahl der Datensätze im Cursor
        int len = dbCursor.getCount();
        final String Touren [][] = new String [len][2];
        String Tourname;
        String Kurzinfo;

        startManagingCursor(dbCursor);

        // bewege Cursor zum ersten Datensatz
        dbCursor.moveToFirst();

        // durchlaufe alle Datensätze im Cursor
        for (int i = 0; i < len; i++) {
            Tourname = dbCursor.getString(NameColumn);
            Kurzinfo = dbCursor.getFloat(WeglaengeColumn) + " km, " +
             dbCursor.getString(DauerColumn) + " h";
            // schreibe Werte in zweidimensionalen Array Touren
            Touren[i][0] = Tourname;
            Touren[i][1] = Kurzinfo;
            // bewege Cursor zum nächsten Datensatz
            dbCursor.moveToNext();
        }

        // übergib Touren an CustomArrayAdapter adap
        CustomArrayAdapter adap=new CustomArrayAdapter(this,
        R.layout.tourenliste, Touren);
        // fülle ListView mit Inhalten aus adap
        setListAdapter(adap);

    }
    finally
        {
            if(db != null) {
                // schließe DB
                dbHelper.close();
            }
        }
}



protected Cursor ListOrderedByName () {
    // DB-Abfrage: sortiere Inhalte der Spalten Tour_Name, Weglaenge_km
    // und Dauer_h aufsteigend  nach Einträgen aus Tour_Name und
    // schreibe sie in Cursor dbCursor
    Cursor dbCursor = db.rawQuery("SELECT Tour_Name, Weglaenge_km,
     Dauer_h FROM " + dbTable + " ORDER BY Tour_Name ASC", null);
```

```
        return dbCursor;
    }



    protected Cursor ListOrderedByDistance () {
         // DB-Abfrage: sortiere Inhalte der Spalten Tour_Name, Weglaenge_km
         // und Dauer_h aufsteigend nach Einträgen aus Weglaenge_km und
         // schreibe sie in Cursor dbCursor
        Cursor dbCursor = db.rawQuery("SELECT Tour_Name, Weglaenge_km,
         Dauer_h FROM " + dbTable + " ORDER BY Weglaenge_km ASC", null);
        return dbCursor;
    }



    @Override
    protected void onListItemClick(ListView l, View v, int position,
    long id) {

            l.setSelector(R.drawable.list_selector);

            super.onListItemClick(l, v, position, id);

            // bei Klick auf Listeneintrag:
            // schreibe Wert der oberen Listenzeile in String-Array tour
            String tour[] =
            (String[])this.getListAdapter().getItem(position);

            // starte Activity Tour.java
            Intent i = new Intent(this, Tour.class);
            startActivity(i);

            // schreibe Namen der ausgewählten Tour in globale Variable
            sel_tour = tour[0];
    }


   // Options-Menü erzeugen
   @Override
   public boolean onPrepareOptionsMenu(Menu menu) {

       menu.clear();

       // wenn Liste alphabetisch sortiert ist
       // füge Eintrag 'Nach Weglänge sortieren' zu Optionsmenü hinzu
       if(sortByAlphabet) {
            menu.removeItem(1);
            menu.add(0, 0, 0, R.string.sort_distance);
        }
       // andernfalls füge 'Alphabetisch sortieren' hinzu
       else
       {
            menu.removeItem(0);
            menu.add(0, 1, 0, R.string.sort_alphabet);
        }
```

```
      getMenuInflater().inflate(R.menu.opt_tourenliste, menu);

      return super.onPrepareOptionsMenu(menu);

    }



  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    // Was wurde im Optionsmenü ausgewählt?
    switch (item.getItemId()) {

        case 1:{
            // sortiere Liste nach Routennamen
            sortByAlphabet = true;
            CreateList();
            return true;
        }

        case 0:{
            // sortiere Liste nach Weglaenge
            sortByAlphabet = false;
            CreateList();
            return true;
        }

        case R.id.opt_beenden: {
            // App beenden
            moveTaskToBack(true);
            return true;
        }

        case R.id.opt_suchen: {
            // rufe CustomSearchDialog auf
            final CustomSearchDialog searchDialog =
            new CustomSearchDialog(this);
            searchDialog.show();
            searchDialog.customSearchDialogBtn.setOnClickListener(
            new View.OnClickListener() {
                public void onClick(View v) {
                        // schreibe eingegebenen Suchbegriff
                        // in globale Variable searchKeywords
                        searchKeywords =
                        searchDialog.customSearchDialogEdit.getText()
                        .toString();
                        searchDialog.hide();
                        // starte Activity Suchergebnis.java
                        Intent i = new Intent(v.getContext(),
                        Suchergebnis.class);
                        startActivity(i);
                    }
                });
            return true;
        }

    }
```

```
  return super.onOptionsItemSelected(item);
}


//farbliche Anpassung des Optionsmenü-Hintergrundes
protected void setMenuBackground(){

      Log.d("OptMenu", "Entering setMenuBackGround");
      getLayoutInflater().setFactory( new Factory() {

          @Override
          public View onCreateView ( String name, Context context,
          AttributeSet attrs ) {

              if(
              name.equalsIgnoreCase(
              "com.android.internal.view.menu.IconMenuItemView")) {

                  try {
                      LayoutInflater f = getLayoutInflater();
                      final View view = f.createView( name, null, attrs );
                      new Handler().post( new Runnable() {
                          public void run () {
                               view.setBackgroundResource(
                               R.drawable.menu_bg_color);
                          }
                      } );
                      return view;
                  }
                  catch ( InflateException e ) {}
                  catch ( ClassNotFoundException e ) {}
              }
              return null;
          }
      });
  }


//'Search'-Knopf des Smartphones belegen
@Override
   public boolean onSearchRequested() {

       // rufe CustomSearchDialog auf
      final CustomSearchDialog searchDialog =
      new CustomSearchDialog(this);
      searchDialog.show();
      searchDialog.customSearchDialogBtn.setOnClickListener(
         new View.OnClickListener() {
          public void onClick(View v) {
                 // schreibe eingegebenen Suchbegriff
                 // in globale Variable searchKeywords
                 TourenListe.searchKeywords =
                 searchDialog.customSearchDialogEdit.getText().toString();
                 searchDialog.hide();
                 // starte Activity Suchergebnis.java
                 Intent i = new Intent(v.getContext(),
                 Suchergebnis.class);
```

```
                    startActivity(i);
                }
            });

        return false;
    }
}
```


ScrollView
RelativeLayout
TableLayout route_infobox
RelativeLayout
RelativeLayout route_infobox_item_0
TextView route_infobox_Wegstrecke
TextView route_infobox_0
RelativeLayout route_infobox_item_1
TextView route_infobox_Anfahrt
TextView route_infobox_1
RelativeLayout route_infobox_item_2
TextView route_infobox_Weglaenge
TextView route_infobox_2
RelativeLayout route_infobox_item_3
TextView route_infobox_Dauer
TextView route_infobox_3
RelativeLayout route_infobox_item_4
TextView route_infobox_Hoehenunterschied
TextView route_infobox_4
RelativeLayout route_rel_layout_5
RelativeLayout route_rel_layout_visible_5
TextView route_etappe_5
RelativeLayout route_rel_layout_btn_5
Button route_btn_5
RelativeLayout route_rel_layout_hidden_5
TextView route_etappe_5_text
TableLayout route_table_5
ImageView route_image_5
RelativeLayout route_rel_layout_6
...
RelativeLayout route_rel_layout_7
...
RelativeLayout route_rel_layout_8
...

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

<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">



<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_marginRight="8sp">


            <TableLayout
                android:id="@+id/route_infobox"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="13sp"
                android:background="@color/Dunkelgruen"
                android:layout_alignParentTop="true">

                <RelativeLayout
                    android:layout_margin="2sp"
                    android:padding="3sp"
                    android:background="@color/Weiss"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content">

                    <RelativeLayout
                       android:id="@+id/route_infobox_item_0"
                       android:layout_width="fill_parent"
                       android:layout_height="wrap_content">

                       <TextView
                           android:id="@+id/route_infobox_Wegstrecke"
                           android:layout_width="wrap_content"
                           android:layout_height="wrap_content"
                           style="@style/Fliesstext_fett"
                           android:layout_alignParentTop="true"
                           android:layout_alignParentLeft="true"
                           android:text="Wegstrecke: " />

                       <TextView
                           android:id="@+id/route_infobox_0"
                           android:layout_width="wrap_content"
                           android:layout_height="wrap_content"
                           style="@style/Fliesstext"
                           android:layout_alignParentTop="true"
                           android:layout_alignParentRight="true"
                           android:layout_toRightOf=
                           "@id/route_infobox_Wegstrecke" />

                   </RelativeLayout>

                   <RelativeLayout
                       android:id="@+id/route_infobox_item_1"
                       android:layout_width="fill_parent"
                       android:layout_height="wrap_content"
                       android:layout_below="@id/route_infobox_item_0">
```

```
        <TextView
            android:id="@+id/route_infobox_Anfahrt"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            style="@style/Fliesstext_fett"
            android:layout_alignParentLeft="true"
            android:text="Anfahrt: " />

        <TextView
            android:id="@+id/route_infobox_1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            style="@style/Fliesstext"
            android:layout_alignParentRight="true"
            android:layout_toRightOf=
           "@id/route_infobox_Anfahrt"/>

    </RelativeLayout>

    <RelativeLayout
        android:id="@+id/route_infobox_item_2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/route_infobox_item_1">

        <TextView
            android:id="@+id/route_infobox_Weglaenge"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            style="@style/Fliesstext_fett"
            android:layout_alignParentLeft="true"
            android:text="Weglänge: " />

        <TextView
            android:id="@+id/route_infobox_2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            style="@style/Fliesstext"
            android:layout_alignParentRight="true"
            android:layout_toRightOf=
            "@id/route_infobox_Weglaenge" />

    </RelativeLayout>

    <RelativeLayout
        android:id="@+id/route_infobox_item_3"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/route_infobox_item_2">

        <TextView
            android:id="@+id/route_infobox_Dauer"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            style="@style/Fliesstext_fett"
            android:layout_alignParentLeft="true"
            android:text="Dauer: " />

        <TextView
            android:id="@+id/route_infobox_3"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
                            style="@style/Fliesstext"
                            android:layout_alignParentRight="true"
                            android:layout_toRightOf=
                           "@id/route_infobox_Dauer" />

                    </RelativeLayout>

                    <RelativeLayout
                        android:id="@+id/route_infobox_item_4"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:layout_below="@id/route_infobox_item_3">

                        <TextView
                            android:id=
                           "@+id/route_infobox_Hoehenunterschied"
                            android:layout_width="wrap_content"
                            android:layout_height="wrap_content"
                            style="@style/Fliesstext_fett"
                            android:layout_alignParentLeft="true"
                            android:text="Höhenunterschied: "/>

                        <TextView
                            android:id="@+id/route_infobox_4"
                            android:layout_width="wrap_content"
                            android:layout_height="wrap_content"
                            style="@style/Fliesstext"
                            android:layout_alignParentRight="true"
                            android:layout_toRightOf=
                            "@id/route_infobox_Hoehenunterschied"/>

                    </RelativeLayout>

                </RelativeLayout>

        </TableLayout>


<!-- route_REL_LAYOUT_5 -->

      <RelativeLayout
            android:id="@+id/route_rel_layout_5"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_marginTop="15sp"
            android:layout_below="@id/route_infobox"
            android:visibility="gone">

            <RelativeLayout
                android:id="@+id/route_rel_layout_visible_5"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_alignParentTop="true"
                android:layout_marginBottom="10sp">

                <RelativeLayout
                    android:id="@+id/route_rel_layout_btn_5"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_below="@id/route_infobox"
                    android:layout_alignParentRight="true">
```

```
                    <Button
                        android:id="@+id/route_btn_5"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_margin="5sp"
                        android:background="@drawable/btn_expander_down"
                        android:layout_gravity="right"/>

                </RelativeLayout>

                <TextView
                    android:id="@+id/route_etappe_5"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    style="@style/Etappe"
                    android:layout_below="@id/route_infobox"
                    android:layout_toLeftOf="@id/route_rel_layout_btn_5"
                    android:layout_alignParentLeft="true"/>

            </RelativeLayout>

            <RelativeLayout
                android:id="@+id/route_rel_layout_5_hidden"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_below="@id/route_rel_layout_visible_5"
                android:visibility="gone">


                <TextView
                    android:id="@+id/route_etappe_5_text"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    style="@style/Fliesstext"
                    android:layout_marginBottom="5sp"
                    android:layout_below="@id/route_rel_layout_visible_5"/>

                <TableLayout
                     android:id="@+id/route_table_5"
                     android:layout_width="wrap_content"
                     android:layout_height="wrap_content"
                     android:background="@color/Schwarz"
                     android:layout_margin="10sp"
                     android:layout_below="@id/route_etappe_5_text">

                    <ImageView
                        android:id="@+id/route_image_5"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:scaleType="fitXY"
                        android:layout_margin="1sp"
                        android:layout_below="@id/route_etappe_5_text"/>

                </TableLayout>

            </RelativeLayout>


        </RelativeLayout>


<!-- route_REL_LAYOUT_6 -->
```

```
<RelativeLayout
    android:id="@+id/route_rel_layout_6"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20sp"
    android:layout_below="@id/route_rel_layout_5"
    android:visibility="gone">

    <RelativeLayout
        android:id="@+id/route_rel_layout_visible_6"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/route_rel_layout_5"
        android:layout_marginBottom="10sp">

        <RelativeLayout
            android:id="@+id/route_rel_layout_btn_6"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/route_rel_layout_5"
            android:layout_alignParentRight="true">

            <Button
                android:id="@+id/route_btn_6"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="5sp"
                android:background="@drawable/btn_expander_down"
                android:layout_gravity="right"/>

        </RelativeLayout>

        <TextView
            android:id="@+id/route_etappe_6"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            style="@style/Etappe"
            android:layout_below="@id/route_rel_layout_5"
            android:layout_toLeftOf="@id/route_rel_layout_btn_6"
            android:layout_alignParentLeft="true"/>

    </RelativeLayout>

    <RelativeLayout
        android:id="@+id/route_rel_layout_6_hidden"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_below="@id/route_rel_layout_visible_6"
        android:visibility="gone">

        <TextView
            android:id="@+id/route_etappe_6_text"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            style="@style/Fliesstext"
            android:layout_marginBottom="5sp"
            android:layout_below="@id/route_rel_layout_visible_6"/>


        <TableLayout
             android:id="@+id/route_table_6"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
```

```
                    android:background="@color/Schwarz"
                    android:layout_margin="10sp"
                    android:layout_below="@id/route_etappe_6_text">

                    <ImageView
                        android:id="@+id/route_image_6"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:scaleType="fitXY"
                        android:layout_margin="1sp"/>

                </TableLayout>

            </RelativeLayout>

        </RelativeLayout>


<!-- route_REL_LAYOUT_7 -->

        <RelativeLayout
            android:id="@+id/route_rel_layout_7"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_marginTop="20sp"
            android:layout_below="@id/route_rel_layout_6"
            android:visibility="gone">

            <RelativeLayout
                android:id="@+id/route_rel_layout_visible_7"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_below="@id/route_rel_layout_6"
                android:layout_marginBottom="10sp">

                <RelativeLayout
                    android:id="@+id/route_rel_layout_btn_7"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_below="@id/route_rel_layout_6"
                    android:layout_alignParentRight="true">

                    <Button
                        android:id="@+id/route_btn_7"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_margin="5sp"
                        android:background="@drawable/btn_expander_down"
                        android:layout_gravity="right"/>

                </RelativeLayout>

                <TextView
                    android:id="@+id/route_etappe_7"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    style="@style/Etappe"
                    android:layout_below="@id/route_rel_layout_6"
                    android:layout_toLeftOf="@id/route_rel_layout_btn_7"
                    android:layout_alignParentLeft="true"/>

            </RelativeLayout>
```

```
            <RelativeLayout
                android:id="@+id/route_rel_layout_7_hidden"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_below="@id/route_rel_layout_visible_7"
                android:visibility="gone">

                <TextView
                    android:id="@+id/route_etappe_7_text"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    style="@style/Fliesstext"
                    android:layout_marginBottom="5sp"
                    android:layout_below="@id/route_rel_layout_visible_7"/>


                <TableLayout
                     android:id="@+id/route_table_7"
                     android:layout_width="wrap_content"
                     android:layout_height="wrap_content"
                     android:background="@color/Schwarz"
                     android:layout_margin="10sp"
                     android:layout_below="@id/route_etappe_7_text">

                    <ImageView
                        android:id="@+id/route_image_7"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:scaleType="fitXY"
                        android:layout_margin="1sp"/>

                </TableLayout>

            </RelativeLayout>

        </RelativeLayout>


<!-- route_REL_LAYOUT_8 -->

        <RelativeLayout
            android:id="@+id/route_rel_layout_8"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_marginTop="20sp"
            android:layout_marginBottom="20sp"
            android:layout_below="@id/route_rel_layout_7"
            android:visibility="gone">

            <RelativeLayout
                android:id="@+id/route_rel_layout_visible_8"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_below="@id/route_rel_layout_7"
                android:layout_marginBottom="10sp">

                <RelativeLayout
                    android:id="@+id/route_rel_layout_btn_8"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_below="@id/route_rel_layout_7"
                    android:layout_alignParentRight="true">
```

```
                    <Button
                        android:id="@+id/route_btn_8"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_margin="5sp"
                        android:background="@drawable/btn_expander_down"
                        android:layout_gravity="right"/>

                </RelativeLayout>

                <TextView
                    android:id="@+id/route_etappe_8"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    style="@style/Etappe"
                    android:layout_below="@id/route_rel_layout_8"
                    android:layout_toLeftOf="@id/route_btn_8"
                    android:layout_alignParentLeft="true"/>

            </RelativeLayout>

            <RelativeLayout
                android:id="@+id/route_rel_layout_8_hidden"
                android:layout_width="wrap_content"
                android:layout_height="fill_parent"
                android:layout_below="@id/route_rel_layout_visible_8"
                android:visibility="gone">

                <TextView
                    android:id="@+id/route_etappe_8_text"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    style="@style/Fliesstext"
                    android:layout_marginBottom="5sp"
                    android:layout_below="@id/route_rel_layout_visible_8"/>


                <TableLayout
                     android:id="@+id/route_table_8"
                     android:layout_width="wrap_content"
                     android:layout_height="wrap_content"
                     android:background="@color/Schwarz"
                     android:layout_margin="10sp"
                     android:layout_below="@id/route_etappe_8_text">

                    <ImageView
                        android:id="@+id/route_image_8"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:scaleType="fitXY"
                        android:layout_margin="1sp"/>

                </TableLayout>

            </RelativeLayout>

        </RelativeLayout>

    </RelativeLayout>

</ScrollView>
```

```
package org.me.wanderfuehrer;

import android.app.Activity;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.os.Handler;
import android.util.AttributeSet;
import android.util.Log;
import android.view.InflateException;
import android.view.LayoutInflater;
import android.view.LayoutInflater.Factory;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.RelativeLayout;
import android.widget.TableLayout;
import android.widget.TextView;

public class Tour_Route extends Activity {

    private static final String dbTable = "Touren";
    protected SQLiteDatabase db;
    private DataBaseHelper dbHelper = new DataBaseHelper(this);

    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);

        // Layout zuweisen
        setContentView(R.layout.tour_route);

        // ändere Hintergrundfarbe des Optionsmenüs
        setMenuBackground();

        final Cursor dbCursor;

        try {

            db = dbHelper.getDataBase();

            // DB-Abfrage: schreibe Inhalte aller Felder des Datensatzes,
            // bei dem Tour_Name mit sel_tour übereinstimmt,
            //in Cursor dbCursor
            dbCursor = db.rawQuery("SELECT * FROM " + dbTable + " WHERE
            Tour_Name = '" + TourenListe.sel_tour  + "'", null);

            startManagingCursor(dbCursor);
            // bewege Cursor zum ersten
            //(und in diesem Falle einzigen) Datensatz
            dbCursor.moveToFirst();

            // laufe von i = 0 bis 8
            // --> IDs in Layoutdatei tour_route.xml von 0 bis 8 nummeriert
```

```
// fülle Views dieses Layouts mit Inhalten aus DB
for (int i = 0; i <= 8; i++) {

    String i_string = (new Integer(i)).toString();

    // i < 5 --> IDs der Infobox-Views
    if (i < 5) {

        // erzeuge Variable für IDs der Infobox
        //in tour_route.xml
        int id_infobox =
        getResources().getIdentifier("route_infobox_"
        + i_string,
        "id", getPackageName());
        TextView infobox = (TextView) findViewById(id_infobox);

        // füge Inhalt der enstprechenden DB-Spalte ein

        if (i == 0) {
                infobox.setText(dbCursor.getString(
                dbCursor.getColumnIndex("Wegstrecke")));
            }
        if (i == 1) {
                infobox.setText(dbCursor.getString(
                dbCursor.getColumnIndex("Anfahrt")));
            }
        if (i == 2) {
                infobox.setText(dbCursor.getString(
                dbCursor.getColumnIndex("Weglaenge_km"))
                + " km");
            }
        if (i == 3) {
                infobox.setText(dbCursor.getString(
                dbCursor.getColumnIndex("Dauer_h")) + " h");
            }
        if (i == 4) {
                infobox.setText(dbCursor.getString(
                dbCursor.getColumnIndex("Hoehenunterschied_m"))
                + " m");
            }

    }
    else {

        // Nummern der DB-Felder entsprechen i-4,
        // da DB-Felder von 1
        // bis 4 nummeriert sind
        String column_nr = (new Integer(i-4)).toString();

        // wenn DB-Spalte 'Etappe_[i-4]_Name' nicht leer,
        // dann mache RelativeLayout für entpr. Etappe sichtbar
        // und fülle es mit Inhalt,
        // andernfalls lasse RelativeLayout unsichtbar
        if (dbCursor.getString(
        dbCursor.getColumnIndex("Etappe_" +
        column_nr + "_Name")) != null) {
```

```
// erzeuge Variable für RelativeLayout-ID
int id_route_rel_layout =
getResources().getIdentifier("route_rel_layout_" +
i_string, "id", getPackageName());
RelativeLayout route_rel_layout =
(RelativeLayout) findViewById(id_route_rel_layout);
// setze RelativeLayout auf sichtbar
route_rel_layout.setVisibility(View.VISIBLE);

// erzeuge Variable für Button-ID
int id_route_btn =
getResources().getIdentifier("route_btn_"
+ i_string, "id", getPackageName());
final Button route_btn =
(Button) findViewById(id_route_btn);

// erzeuge Variable für Etappen-ID
int id_route_etappe =
getResources().getIdentifier("route_etappe_" +
i_string, "id", getPackageName());
TextView route_etappe =
(TextView) findViewById(id_route_etappe);

// füge Inhalt der DB-Spalte 'Etappe_[i-4]_Name'
// und 'Etappe_[i-4]_Dauer_min' ein
route_etappe.setText(dbCursor.getString(
dbCursor.getColumnIndex("Etappe_" + column_nr
+ "_Name"))
+ " ["
+ dbCursor.getString(dbCursor.getColumnIndex
("Etappe_" + column_nr + "_Dauer_min"))
+ " min]");

// erzeuge Variable für RelativeLayout-ID
int id_route_rel_layout_hidden =
getResources().getIdentifier("route_rel_layout_" +
i_string + "_hidden", "id", getPackageName());
final RelativeLayout route_rel_layout_hidden =
(RelativeLayout) findViewById
(id_route_rel_layout_hidden);
route_rel_layout_hidden.setVisibility(View.GONE);

// erzeuge OnClick-Listener für alle Buttons
route_btn.setOnClickListener(
new View.OnClickListener() {

    public void onClick(View arg0) {
        // wenn route_rel_layout_hidden unsichtbar:
        // ändere Button-Hintergrund und
        //setze RelativeLayout auf sichtbar
        if (route_rel_layout_hidden.getVisibility()
         == View.GONE) {
            route_rel_layout_hidden
            .setVisibility(View.VISIBLE);
            route_btn
            .setBackgroundResource(
           R.drawable.btn_expander_up);
```

```
            }
            // andernfalls':
            // ändere Button-Hintergrund und
            // setze RelativeLayout auf unsichtbar
            else {
                  route_rel_layout_hidden
                  .setVisibility(View.GONE);
                  route_btn
                  .setBackgroundResource(
                  R.drawable.btn_expander_down);
            }
        }
    });

    // erzeuge Variable für Etappen-Text-ID
    int id_route_etappe_text =
    getResources().getIdentifier("route_etappe_" +
    i_string + "_text", "id", getPackageName());
    TextView route_etappe_text =
    (TextView) findViewById(id_route_etappe_text);

    // füge Inhalt der DB-Spalte
    // 'Etappe_[i-4]_Text' ein
    route_etappe_text.setText(
    dbCursor.getString(
    dbCursor.getColumnIndex("Etappe_" +
    column_nr + "_Text")));


    // entnimm Bild aus DB
    byte[] b =
    dbCursor.getBlob(dbCursor.getColumnIndex("Etappe_"
    + column_nr + "_Bild"));

    // wenn DB-Feld leer, also kein Bild vorhanden ist,
    // setze TableLayout auf unsichtbar
    //(sonst Rahmen sichtbar)
    if (b == null) {
        // erzeuge Variable für Table-ID
        int id_table =
        getResources().getIdentifier("route_table_" +
        i_string, "id", getPackageName());
        TableLayout table =
        (TableLayout) findViewById(id_table);
        table.setVisibility(View.GONE);
    }
    else {
        // andernfalls entnimm Bild aus DB
        //und dekodiere es
        Bitmap bitmap =
        BitmapFactory.decodeByteArray(b, 0, b.length);

        // erzeuge Variable für Image-ID
        int id_etappe_image =
        getResources().getIdentifier("route_image_" +
        i_string, "id", getPackageName());
        ImageView etappe_image =
```

```
                            (ImageView) findViewById (id_etappe_image);
                            // füge Bitmap in ImageView ein
                            etappe_image.setImageBitmap(bitmap);

                        }
                    }
                }
            }
        }
       finally
           {
               if(db != null) {
                   // schließe DB
                   dbHelper.close();
               }
           }

    }


   // farbliche Anpassung des Optionsmenü-Hintergrundes
   protected void setMenuBackground(){

       Log.d("OptMenu", "Entering setMenuBackGround");
       getLayoutInflater().setFactory( new Factory() {

           @Override
           public View onCreateView ( String name, Context context,
           AttributeSet attrs ) {

               if(
               name.equalsIgnoreCase(
               "com.android.internal.view.menu.IconMenuItemView" ) ) {

                   try {
                       LayoutInflater f = getLayoutInflater();
                       final View view = f.createView( name, null, attrs );
                       new Handler().post( new Runnable() {
                           public void run () {
                                view.setBackgroundResource(
                                R.drawable.menu_bg_color);
                           }
                       } );
                       return view;
                   }
                   catch ( InflateException e ) {}
                   catch ( ClassNotFoundException e ) {}
               }
               return null;
           }
       });
   }
}
```

```
package org.me.wanderfuehrer;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.RectF;

import org.osmdroid.util.GeoPoint;
import org.osmdroid.views.MapView;
import org.osmdroid.views.overlay.Overlay;
import org.osmdroid.views.MapView.Projection;

public class CustomOverlay extends Overlay
{
      private GeoPoint gp1;
      private GeoPoint gp2;
      private int mRadius=6;
      private int mode=0;
      private int defaultColor;
      String text= "";
      Bitmap img = null;


      // Konstruktor
      public CustomOverlay (Context context) {
            super(context);
      }

      // definiere Punkte, die verbunden werden sollen
      public void setPoints (GeoPoint gp1, GeoPoint gp2, int mode) {
          this.gp1 = gp1;
          this.gp2 = gp2;
          this.mode = mode;
          defaultColor = 999; // no defaultColor
      }

      // definiere Punkte, die verbunden werden sollen und Farbe,
      // in der gezeichnet werden soll
      public void setPointsAndColor (GeoPoint gp1, GeoPoint gp2, int mode,
      int defaultColor) {
          this.gp1 = gp1;
          this.gp2 = gp2;
          this.mode = mode;
          this.defaultColor = defaultColor;
      }

      // zeichnet Pfad
      @Override
      public void onDraw (Canvas canvas, MapView mapView)
      {
            Projection projection = mapView.getProjection();

            Paint paint = new Paint();
            paint.setAntiAlias(true);
```

```
Point point = new Point();
projection.toPixels(gp1, point);

// mode = 1 --> Startpunkt des zu zeichnenden Pfades
if(mode == 1) {

     if(defaultColor == 999) {
          paint.setColor(Color.RED);
     }
     else {
          paint.setColor(defaultColor);
     }

     RectF oval=new RectF(point.x - mRadius, point.y - mRadius,
     point.x + mRadius, point.y + mRadius);
     // zeichne Kreis
     canvas.drawOval(oval, paint);
}

// mode = 2 --> Pfad
else if(mode == 2) {

     if(defaultColor == 999) {
          paint.setColor(Color.RED);
     }
     else {
          paint.setColor(defaultColor);
     }

     Point point2 = new Point();
     projection.toPixels(gp2, point2);
     paint.setStrokeWidth(3);
     paint.setAlpha(200);
     // zeichne Linie zwischen Punkten
    canvas.drawLine(point.x, point.y, point2.x,point2.y, paint);
     }

// mode = 3 --> Endpunkt des Pfades
else if(mode == 3) {

     if(defaultColor == 999) {
          paint.setColor(Color.RED);
     }
     else  {
          paint.setColor(defaultColor);
     }

     Point point2 = new Point();
     projection.toPixels(gp2, point2);
     paint.setStrokeWidth(5);
     paint.setAlpha(120);
    canvas.drawLine(point.x, point.y, point2.x,point2.y, paint);
     RectF oval=new RectF(point2.x - mRadius,point2.y - mRadius,
     point2.x + mRadius,point2.y + mRadius);
     // zeichne Kreis
     paint.setAlpha(255);
     canvas.drawOval(oval, paint);
```

```
            }
        }
        @Override
        protected void onDrawFinished (final Canvas c, final MapView osmv) {

        }
}
```

```
package org.me.wanderfuehrer;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.util.ArrayList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.osmdroid.DefaultResourceProxyImpl;
import org.osmdroid.tileprovider.tilesource.TileSourceFactory;
import org.osmdroid.util.BoundingBoxE6;
import org.osmdroid.util.GeoPoint;
import org.osmdroid.views.MapController;
import org.osmdroid.views.MapView;
import org.osmdroid.views.overlay.ItemizedOverlay;
import org.osmdroid.views.overlay.OverlayItem;
import org.osmdroid.views.overlay.ScaleBarOverlay;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import android.app.Activity;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Color;
import android.graphics.Point;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.AttributeSet;
import android.util.Log;
import android.view.InflateException;
import android.view.LayoutInflater;
import android.view.LayoutInflater.Factory;
import android.view.Menu;
import android.view.View;
import android.widget.Button;

public class Tour_Karte extends Activity implements LocationListener {

	private MapView mapView;
	private MapController mapController;
	private ScaleBarOverlay mScaleBarOverlay;
	private CustomSimpleLocationOverlay curPosOverlay;

	public static LocationManager locationManager;
	public static LocationListener locationListener;
```

```
  public static boolean requestLocation = false;
  public static Button btnUserPos;

  private static final String dbTable = "Touren";
  protected SQLiteDatabase db;
  private final DataBaseHelper dbHelper = new DataBaseHelper(this);

  final ArrayList<GeoPoint> bbList = new ArrayList<GeoPoint>();

  private boolean finish = false;
  private boolean data;
  private CustomProgressDialog progressDialog;

@Override
public void onCreate(Bundle icicle) {
     super.onCreate(icicle);
     // Layout zuweisen
     setContentView(R.layout.tour_karte);

     // ändere Hintergrundfarbe des Optionsmenüs
     setMenuBackground();

      this.mapView =
      (MapView) this.findViewById(R.id.tour_karte_mapview);
     this.mapView.setTileSource(TileSourceFactory.MAPNIK);
     this.mapController = this.mapView.getController();
     this.mapController.setZoom(1);


      // Zoom-In-Button
     Button btnZoomIn =
    (Button) this.findViewById(R.id.tour_karte_btn_zoom_in);
     btnZoomIn.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
             mapController.zoomIn();
         }

     });

     // Zoom-Out-Button
     Button btnZoomOut =
    (Button) this.findViewById(R.id.tour_karte_btn_zoom_out);
     btnZoomOut.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
             mapController.zoomOut();
         }

     });

     // Locate-User-Button
     btnUserPos =
    (Button) this.findViewById(R.id.tour_karte_btn_user_pos);
     btnUserPos.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {

              // wenn boolean-Variable requestLocation = false
              if (!requestLocation) {
```

```
            locationManager = (LocationManager)
            Tour_Karte.this.getSystemService(
            Context.LOCATION_SERVICE);

            // wenn GPS auf Smartphone deaktiviert ist:
            // zeige Dialog mit Info
            if(!locationManager.isProviderEnabled(
            LocationManager.GPS_PROVIDER ))
            {
                  final CustomDialog GPSDialog =
                  new CustomDialog(Tour_Karte.this);
                  GPSDialog.show();
                  GPSDialog.customDialogBtn.setOnClickListener(
                  new View.OnClickListener() {
                   public void onClick(View v) {
                        GPSDialog.cancel();
                       }
                   });
                  GPSDialog.customDialogTitle.setText(
                  R.string.gps_settings);
                  GPSDialog.customDialogText.setText(
                  R.string.gps_disabled);
            }
            // andernfalls: starte Lokalisierung
            else {

                  requestLocation = true;

                  // ändere Button-Hintergrund
                  btnUserPos.setBackgroundResource(
                  R.drawable.btn_disable_cur_pos);

                  locationListener = Tour_Karte.this;

                  // aktualisiere Nutzer-Position aller 5000ms
                  // bzw. 100m
                  locationManager.requestLocationUpdates(
                  LocationManager.GPS_PROVIDER, 5000, 100,
                  locationListener);

                 // füge CustomSimpleLocationOverlay
                  // curPosOverlay zu Karte hinzu
                 Tour_Karte.this.curPosOverlay =
                new  CustomSimpleLocationOverlay(Tour_Karte.this);
                  Tour_Karte.this.mapView.getOverlays()
                  .add(curPosOverlay);

            }
      }
      // andernfalls: stoppe Lokalisierung
      else {
            requestLocation = false;

            // ändere Button-Hintergrund
            btnUserPos.setBackgroundResource(
            R.drawable.btn_cur_pos);
```

```
                    // entferne SimpleLocationOverlay mit Nutzer-Position
                    Tour_Karte.this.mapView.getOverlays()
                    .remove(curPosOverlay);
                    Tour_Karte.this.mapView.invalidate();

                    // stoppe Lokalisierung
                    locationManager.removeUpdates(Tour_Karte.this);
               }

          }

     });


     progressDialog = new CustomProgressDialog(this);
     progressDialog.show();
     progressDialog.customProgressDialogText.setText(
      R.string.loading_route);

     // zeige ProgressDialog während Routen aus DB gelesen werden
     new Thread() {
        public void run() {

               // führe DrawRoute() aus, solange finish = false
               while(!finish){
                    // zeichne Route
                    DrawRoute(Color.RED, mapView);

                    // erzeuge und setze Maßstabsleiste für Karte
                    // (wird zum Schluss gemacht,
                    // damit Maßstableiste der oberste Overlay ist)
                    Tour_Karte.this.mScaleBarOverlay =
                    new ScaleBarOverlay(Tour_Karte.this);
                    Tour_Karte.this.mScaleBarOverlay
                    .setScaleBarOffset(10, 320);
                    Tour_Karte.this.mapView.getOverlays()
                    .add(mScaleBarOverlay);
               }
               handler.sendEmptyMessage(0);
        }
     }.start();


}


private Handler handler = new Handler() {
       @Override
       public void handleMessage(Message msg) {

               // schließe ProgressDialog
               progressDialog.dismiss();

               // wenn kml-Daten zu Route vorhanden:
               if(data) {
                  // zoome an Route heran
```

```
                Tour_Karte.this.mapController.setZoom(13);
                // berechne BoundingBox der GeopPoint-ArrayList bbList
                BoundingBoxE6 bb = BoundingBoxE6.fromGeoPoints(bbList);
                // setze Mittelpunkt der BoundingBox
                //als Kartenmittelpunkt
                Tour_Karte.this.mapController.setCenter(bb.getCenter());
                Log.d("MapView", "Center of BoundingBox: " +
                bb.getCenter().getLatitudeE6() + "   " +
                bb.getCenter().getLongitudeE6());
            }
            // andernfalls: zeige Dialog mit Info
            else {
             final CustomDialog MapDialog =
             new CustomDialog(Tour_Karte.this);
             MapDialog.show();
             MapDialog.customDialogBtn.setOnClickListener(
             new View.OnClickListener() {
                 public void onClick(View v) {
                         MapDialog.dismiss();
                     }
                 });
             MapDialog.customDialogTitle.setText(R.string.sorry);
             MapDialog.customDialogText.setText(R.string.no_data);
          }
       }
 };

 // unterbreche das Laden der Route
 public void finishLoadRoute() {
       finish = true;
 }

 // Nutzer-Position hat sich verändert
 public void onLocationChanged(Location location) {

    if (location != null) {

       // erzeuge GeoPoint mit aktueller Nutzer-Position
       GeoPoint curPos =
       new GeoPoint (location.getLatitude(), location.getLongitude());

       // zeige aktuele Nutzer-Position als Positionssignatur
       // im SimpleLocationOverlay curPosOverlay
       Tour_Karte.this.curPosOverlay.setLocation(curPos);
       // bewege Kartenausschnitt zu aktueller Nutzerposition
       Tour_Karte.this.mapController.animateTo(curPos);
       Log.w("MapView", "current User Position: "
       + curPos.getLatitudeE6()
       + "   " + curPos.getLongitudeE6());
 }

}

public void onProviderDisabled(String provider) {
}

public void onProviderEnabled(String provider) {
```

```
    }

    public void onStatusChanged(String provider, int status, Bundle extras)
{
    }

    protected boolean isRouteDisplayed() {
      return false;
    }


    // bei Beenden der Activity
    @Override
    public void onDestroy() {
      // wenn GPS noch aktiv ist:
      // stoppe Lokalisierung
      if (requestLocation) {
            locationManager.removeUpdates(Tour_Karte.this);
            requestLocation = false;
      }
      super.onDestroy();
    }



     // Route in Karte zeichnen
     private void DrawRoute(int color, MapView mMapView01) {

      Document doc = null;

      try
      {
          final Cursor dbCursor;

          final ArrayList<OverlayItem> placemarks =
          new ArrayList<OverlayItem>();

          // öffne DB
          db = dbHelper.getDataBase();
          // DB-Abfrage: schreibe Inhalt des DB-Feldes KML des Datensatzes,
          // bei dem Tour_Name mit sel_tour übereinstimmt,
          // in Cursor dbCursor
          dbCursor = db.rawQuery("SELECT KML FROM " + dbTable + " WHERE
          Tour_Name = '" + TourenListe.sel_tour + "'", null);

          startManagingCursor(dbCursor);
          // bewge Cursor zu erstem Datensatz
          dbCursor.moveToFirst();
          // Inhalt des Cursors
          byte[] b = dbCursor.getBlob(dbCursor.getColumnIndex("KML"));

          // wenn kml-Daten vorliegen
          if (b != null) {

            // schreibe kml-Daten in InputStream
            InputStream is = new ByteArrayInputStream(b);
```

```
DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
DocumentBuilder dbb = dbf.newDocumentBuilder();
// erzeuge Document aus InputStream
doc = dbb.parse(is);
doc.getDocumentElement().normalize();

// erzeuge NodeList mit allen LineString-Tags des Documents
NodeList LineStringNodeList =
doc.getElementsByTagName("LineString");
Node LineStringNode = LineStringNodeList.item(0);
Element LineStringElement = (Element) LineStringNode;
// erzeuge NodeList mit allen coordinate-Tags der
// LineStringNodeList
NodeList CoordList =
LineStringElement.getElementsByTagName("coordinates");
Element CoordElement = (Element) CoordList.item(0);
// schreibe Inhalte der Knoten aus CoordList in String path
String path = CoordElement.getFirstChild().getNodeValue();

// splitte path bei auftretenden Leerzeichen auf
// und schreibe jeden Teil in String-Array pairs
// pairs enthält nun alle Koordinaten-Paare aus LineString
String [] pairs = path.split(" ");
// splitte pairs bei auftretenden Kommas auf, um einzelne
// Koordinatenwerte zu erhalten
// --> LngLatLineString[0]=Länge, LngLatLineString[1]=Breite,
// LngLatLineString[2]=Höhe
String[] LngLatLineString = pairs[0].split(",");

// schreibe Länge und Breite der ersten Koordinate
// in GeoPoint startGP
// --> Startpunkt der jeweiligen Route
GeoPoint startGP = new GeoPoint((int)
(Double.parseDouble(LngLatLineString[1])*1E6),
(int)(Double.parseDouble(LngLatLineString[0])*1E6));
// füge der Karte ein Overlay mit Startpunkt hinzu
CustomOverlay mMyOverLay1 = new CustomOverlay (this);
mMyOverLay1.setPoints(startGP, startGP, 1);
mMapView01.getOverlays().add(mMyOverLay1);

GeoPoint gp1;
GeoPoint gp2 = startGP;
// füge Startpunkt zu ArrayList für Berechnung
// der BoundingBox hinzu
bbList.add(gp2);

// durchlaufe restliche Koordinaten
for(int i=1;i<pairs.length;i++)
{
// wenn es sich nicht um letzten Punkt handelt:
if (i != (pairs.length - 1)) {
      // splitte pairs bei auftretenden Kommas auf, um einzelne
      // Koordinatenwerte zu erhalten
      // --> LngLatLineString [0]=Länge,
      // LngLatLineString [1]=Breite, LngLatLineString t[2]=Höhe
      // LngLatLineString = pairs[i].split(",");
```

```
                gp1 = gp2;
                gp2 = new GeoPoint((int)
                (Double.parseDouble(LngLatLineString[1])*1E6),
                (int)(Double.parseDouble(LngLatLineString[0])*1E6));

                // füge der Karte ein Overlay mit Verbindungslinie
                //zwischen letztem und aktuellen Punkt hinzu
                CustomOverlay mMyOverLay2 = new CustomOverlay (this);
                mMyOverLay2.setPointsAndColor(gp1, gp2, 2, color);
                mMapView01.getOverlays().add(mMyOverLay2);
                }
        // andernfalls:
        else {
              // splitte pairs bei auftretenden Kommas auf, um einzelne
              // Koordinatenwerte zu erhalten
              // --> LngLatLineString [0]=Länge,
              // LngLatLineString [1]=Breite, LngLatLineString [2]=Höhe
              LngLatLineString = pairs[i].split(",");
              gp1 = gp2;
              gp2 =
              new GeoPoint((int)
              (Double.parseDouble(LngLatLineString[1])*1E6),
              (int)(Double.parseDouble(LngLatLineString[0])*1E6));

              // füge der Karte ein Overlay mit Verbindungslinie
              // (zwischen letztem Punkt und Endpunkt)
              // und Endpunkt hinzu
              CustomOverlay mMyOverLay3 = new CustomOverlay (this);
              mMyOverLay3.setPoints(gp2, gp2, 3);
              mMapView01.getOverlays().add(mMyOverLay3);
        }
        Log.d("LineString-Coordinates","pair:" + pairs[i]);

        // füge aktuellen Punkt zu ArrayList für Berechnung
        // der BoundingBox hinzu
        bbList.add(gp2);
      }


// erzeuge NodeList mit allen Point-Tags des Documents
NodeList PointNodeList = doc.getElementsByTagName("Point");

// durchlaufe PointNodeList
for (int i=0; i < PointNodeList.getLength(); i++) {

      Node PointNode = PointNodeList.item(i);
      Element PointElement = (Element) PointNode;
      // erzeuge NodeList mit allen coordinate-Tags der PointNodeList
      NodeList PointCoordList =
      PointElement.getElementsByTagName("coordinates");
      Element PointCoordElement = (Element) PointCoordList.item(0);
      // schreibe Inhalt des Knotens aus PointCoordElement
      // in String pair
      String pair = PointCoordElement.getFirstChild().getNodeValue();

      // splitte pair bei auftretenden Kommas auf,
```

```
// um einzelne Koordinatenwerte zu erhalten
// --> LngLatPoint[0]=Länge, LngLatPoint[1]=Breite,
// LngLatPoint[2]=Höhe
String[] LngLatPoint = pair.split(",");

// schreibe Länge und Breite der Koordinate
// in GeoPoint PlacemarkGP
GeoPoint PlacemarkGP =
new GeoPoint((int)(Double.parseDouble(LngLatPoint[1])*1E6),
(int)(Double.parseDouble(LngLatPoint[0])*1E6));

// greife auf Eltern-Knoten der PointNodeList zu
Node PlacemarkNode = PointNodeList.item(i).getParentNode();
Element PlacemarkElement = (Element) PlacemarkNode;

// erzeuge NodeList mit allen name-Tags von PlacemarkElement
NodeList NameList =
PlacemarkElement.getElementsByTagName("name");
Element NameElement = (Element) NameList.item(0);
String PlacemarkName = null;

// versuche, auf Inhalt des name-Tags zuzugreifen
// und in String PlacemarkName zu schreiben
try {
     PlacemarkName = NameElement.getFirstChild().getNodeValue();
}

// falls NameElement leer ist: fange NullPointerException ab,
// sonst Error
catch(NullPointerException e)
{
     e.printStackTrace();
}

// wenn NameElement bzw. PlacemarkName nicht leer ist:
if(PlacemarkName != null) {
     PlacemarkName = NameElement.getFirstChild().getNodeValue();
    Log.d("kml-Placemark", "Placemark-Name = " + PlacemarkName);

     // erzeuge NodeList mit allen discription-Tags
     // von PlacemarkElement
     NodeList DescriptionList =
     PlacemarkElement.getElementsByTagName("description");
     Element DescriptionElement =
     (Element) DescriptionList.item(0);
     String PlacemarkDescription;

     // wenn DescritpionList die Länge 0 hat also leer ist:
     if(DescriptionList.getLength() == 0) {
          PlacemarkDescription = " ";
     }
     // andernfalls:
     else {
          // schreibe Inhalt des Knoten
          // in String PlacemarkDescription
          PlacemarkDescription =
          DescriptionElement.getFirstChild().getNodeValue();
```

```
                Log.d("kml-Placemark", "PlacemarkDescription = " +
                PlacemarkDescription);
            }

            // füge GeoPoint mit Koordinaten, Namen und Beschreibung
            // des aktuellen Punktes zu ArrayList placemarks hinzu
            placemarks.add(new OverlayItem(PlacemarkName,
            PlacemarkDescription, PlacemarkGP));
            Log.d("Point-Coordinates","pair:" + pair);

            // füge aktuellen Punkt zu ArrayList für Berechnung der
            // BoundingBox hinzu
            bbList.add(PlacemarkGP);
     }

    }

            // erzeuge CustomItemizedOverlayWithFocus mit Markern für
            // Placemarks der Routen
            CustomItemizedOverlayWithFocus<OverlayItem> pmOverlay;
            pmOverlay =
            new CustomItemizedOverlayWithFocus<OverlayItem> (
                  this,
                  placemarks,
                  getResources().getDrawable(R.drawable.map_marker),
                  new Point (13, 46),
                  getResources().getDrawable(
                  R.drawable.map_marker_focused_base),
                  new Point (10, 16),
                  Color.parseColor("#FFFFE6"),
                  new
                  ItemizedOverlay.OnItemGestureListener<OverlayItem>(){

                  @Override
                      public boolean onItemSingleTapUp(final int index,
                       final OverlayItem item) {
                            return true;
                       }

                       @Override
                       public boolean onItemLongPress(final int index,
                       final OverlayItem item) {
                              return false;
                       }

    }, new DefaultResourceProxyImpl(getApplicationContext()));

     pmOverlay.setFocusItemsOnTap(true);

     mMapView01.getOverlays().add(pmOverlay);

     // kml-Daten zu Route vorhanden
     data = true;
     // Ladevorgang beenden
     finishLoadRoute();

    }
```

```
        // andernfalls:
         else {
               // keine kml-Daten zu Route vorhanden
               data = false;
               // Ladevorgang beenden
               finishLoadRoute();
         }

         }
         catch (MalformedURLException e)
         {
               e.printStackTrace();
         }

         catch (IOException e)
         {
               e.printStackTrace();
         }

         catch (ParserConfigurationException e)
         {
               e.printStackTrace();
         }

         catch (SAXException e)
         {
               e.printStackTrace();
         }
         finally
        {
            if(db != null) {
                // DB schließen
                dbHelper.close();
            }
        }
    }

 // Options-Menü erzeugen
 @Override
 public boolean onPrepareOptionsMenu(Menu menu) {

     menu.clear();
     menu.add(0, 0, 0, R.string.legend);
     getMenuInflater().inflate(R.menu.opt_tour, menu);

     return super.onPrepareOptionsMenu(menu);

  }

// farbliche Anpassung des Optionsmenü-Hintergrundes
protected void setMenuBackground(){

    Log.d("OptMenu", "Entering setMenuBackGround");
    getLayoutInflater().setFactory( new Factory() {

        @Override
        public View onCreateView ( String name, Context context,
```

```
            AttributeSet attrs ) {

                if (
                 name.equalsIgnoreCase(
                 "com.android.internal.view.menu.IconMenuItemView" )) {

                    try {
                        LayoutInflater f = getLayoutInflater();
                        final View view = f.createView( name, null, attrs );
                        new Handler().post( new Runnable() {
                            public void run () {
                                view.setBackgroundResource(
                                R.drawable.menu_bg_color);
                            }
                        } );
                        return view;
                    }
                    catch ( InflateException e ) {}
                    catch ( ClassNotFoundException e ) {}
                }
                return null;
            }
        });
    }
}
```

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

<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:top="63dp">

        <shape android:shape="rectangle">
            <solid android:color="@color/Olivgruen" />
        </shape>

    </item>

    <item android:left="2dp" android:right="2dp">

        <shape android:shape="rectangle">
            <corners
                android:bottomLeftRadius="0.1dp"
                android:bottomRightRadius="0.1dp"
                android:topLeftRadius="4dp"
                android:topRightRadius="4dp" />

            <gradient
                android:angle="90"
                android:startColor="@color/Hellgruen"
                android:centerColor="@color/Olivgruen"
                android:endColor="@color/Hellgruen"/>

            <stroke
                android:width="1px"
                android:color="@color/Olivgruen" />
        </shape>

    </item>

</layer-list>
```

Repräsentationsfunktion des Icons	**Icons der Applikation ‚Wandern in der Sächischen Schweiz'**	**Standard-Icons von Android 1.5 bzw. osmdroid 3.0.2**
Suchfunktion		
Tab *Tour_Route.java*		
Tab *Tour_Karte.java*		
Tab *Tour_Info.java*		
Aufblättern eines Fließtextes		
Verbergen eines Fließtextes		—
Reinzoomen in Karte		Android 1.5:  osmdroid 3.0.2: 
Rauszoomen aus Karte		Android 1.5:  osmdroid 3.0.2: 
Nutzerlokalisierung starten		
Nutzerlokalisierung beenden		—
Standort des Benutzers in Karte		
POI in Karte		
Basis eines fokussierten POI in Karte		

Danksagung

An erster Stelle möchte ich ganz herzlich Herrn Prof. Dr.-Ing. Dirk Burghardt für seine Unterstützung und sein ständiges Interesse danken, sowohl im Hinblick auf diese Diplomarbeit als auch darüber hinaus, was in einer fortführenden Zusammenarbeit resultiert. Vielen, vielen Dank dafür.

Ebenso geht ein großes Dankeschön an Peggy Thiemt, die mir bei den Programmierarbeiten sehr hilfreich und produktiv mit Rat und Tat zur Seite stand sowie an Stefan Hahmann für wertvolle Hinweise und Zusatzinformationen für den Inhalt meiner Diplomarbeit.

Des Weiteren bedanke ich mich bei meinen Eltern und meiner Stiefmutter für ihre nicht abreißende Unterstützung, ihr ständiges Interesse und ihr Vertrauen in mich; bei Helen, Christian, Elena, Elli, Norbert, Josi, Matthi und Giso für die wohltuenden, ablenkenden und unbeschwerten gemeinsamen Abende sowie bei Laila Nahara für das tolle Training, das mir ausnahmslos Freude und Kraft gespendet und mir jede Woche erneut gezeigt hat, dass ich nicht nur aus einem Kopf bestehe.

Und nicht zu vergessen: mein Kater Ziegel, der mich trotz ‚Sitzblockaden' auf meinem Schoß, dem Schreibtisch oder gar der Laptop-Tastatur, mit seiner herzerfrischenden Art und Weise jeden Tag aufs Neue zum Lachen gebracht hat.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage der Diplomkommission der Fachrichtung Geowissenschaften eingereichte Diplomarbeit zum Thema

„Analyse, Konzeption und Entwicklung einer mobilen Kartenanwendung auf Basis des Wanderkalenders der Sächsischen Zeitung“

vollkommen selbständig und nur unter Benutzung der in der Arbeit angegebenen Literatur angefertigt habe.

Pirna, den 28. April 2011 .....

Eva Hauthal