Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3108

# Entwicklung eines Frontend-Generators für Testanwendungen eines Informationssystems

Markus Knittig

| | |
|---|---|
| **Studiengang:** | Softwaretechnik |
| **Prüfer:** | Prof. Dr. Thomas Ertl |
| **Betreuer:** | Dipl.-Inf. Florian Haag |
| **begonnen am:** | 17. November 2010 |
| **beendet am:** | 17. Mai 2011 |
| **CR-Klassifikation:** | D.1.2, D.2.11, I.6.5, H.5.2 |

# Inhaltsverzeichnis

# 1. Einleitung

Die Reise- und Verbindungsplanung im öffentlichen Verkehr ist meist immer noch relativ zeitaufwendig für den Fahrgast. Es fehlt an Standardisierung und die Systeme beschränken sich größtenteils auf die klassische Verbindungssuche. Im Rahmen des Forschungsprojektes IP-KOM-ÖV [IPK] soll unter anderem der Zugang zu Fahrgastinformationen verbessert werden und so die Basis für neue Informationsdienste für den Fahrgast geschaffen werden [IPK10].

Ein wesentlicher Baustein sind dabei mobile Geräte. Damit sind insbesondere Smartphones gemeint, aber auch Tablets entwickeln sich immer mehr vom Nischenprodukt zum Massenprodukt. Ein Grundstein des Erfolges sind dabei die verfügbaren Anwendungen, sogenannte Apps, auf mobilen Geräten. Die Hardware von Smartphones ist mittlerweile stark genug, um innovative Anwendungen aller Art zu ermöglichen.

Im Rahmen des Projektes IP-KOM-ÖV sollen Prototypen erstellt werden, die verschiedene Szenarien von Fahrgastinformationssystemen auf einem mobilen Gerät in Form von Apps abbilden sollen.

Dazu soll modellgetriebende Softwareentwicklung eingesetzt werden. Statt die Prototypen von Hand zu implementieren, soll ein formales Modell erstellt werden, das die Anforderungen der Verkehrsinformationsdomäne beschreibt. Mit Hilfe von diesem formalen Modell sollen dann Prototypen generiert werden, anstatt diese manuell zu implementieren. Damit wird der Aufwand zur Erstellung neuer Prototypen reduziert, sowie eine einfachere Entwicklung für Fachanwender und Entwickler gewährleistet.

Ziel ist es im Rahmen dieser Diplomarbeit mit Hilfe von modellgetriebener Softwareentwicklung einen Generator für Prototypen für Verkehrsinformationssysteme zu schreiben, der im Rahmen des Projektes IP-KOM-ÖV eingesetzt werden kann.

In der Praxis konnte sich die modellgetriebene Softwareentwicklung allerdings noch nicht in der Breite durchsetzen. Insbesondere für grafische Benutzeroberflächen fehlt es hier an praktikablen Lösungen [DKW10]. Ein sekundäres Ziel ist es daher, im Rahmen dieser Diplomarbeit festzustellen, wie praktikabel die modellgetriebende Softwareentwicklung sich für eine bestimmte Anwenderdomäne einsetzen lässt.

## 1.1. Aufbau

Im Kapitel 2 werden die Anforderungen an den Prototyp Generator detailliert beschrieben. Im Kapitel 3 werden die Grundlagen der Arbeit vorgestellt. Hier erfolgt die Evaluation

und Auswahl der verwendeten Plattformen und Ansätze. In Kapitel 4 wird die Konzeption des Prototyp Generators unabhängig von konkret eingesetzten Technologien erläutert. Im Kapitel 5 wird dann der technische Entwurf und die Realisierung des Prototyps beschrieben. Abschließend werden in der Zusammenfassung die Ergebnisse der Arbeit bewertet und im Ausblick auf zukünftige Entwicklungen eingegangen.

# 2. Aufgabenstellung

In diesem Kapitel soll die grundsätzliche Aufgabenstellung der Diplomarbeit erläutert werden. Dazu werden die Funktion, die diese Arbeit im Rahmen des IP-KOM-ÖV Projektes erklärt, sowie Anforderungen aus der Ausschreibung und Anfangsphase der Diplomarbeit beschrieben.

Ausgangspunkt im Projekt IP-KOM-ÖV ist die Gesamtvorhabensbeschreibung, die Teil des Forschungsantrages ist [IPK10]. In diesem Dokument werden unter anderem Arbeitspakete für die einzelnen Projektpartner beschrieben. Diese Arbeit soll zur Fertigstellung von Arbeitspaketen der Universität Stuttgart beitragen. Die betreffenden Arbeitspakete sind Teil des Arbeitskomplex 2 "Kommunikationsdienste für Kundengeräte". Das Arbeitspaket 2.100 „Vorstudie und Funktionale Beschreibung" wurde bereits in der Fachstudie „IT-Einsatzszenarien zur interaktiven Fahrgastunterstützung" bearbeitet [AWZ10] und dient als einer der Ausgangspunkte der Arbeit. Die Diplomarbeit ist Teil von Arbeitspaket 2.300 „Machbarkeitsnachweis und Prototyping für Interaktion, Mobilgeräte und Service-Orientierung". Ein Ziel des Arbeitspaketes ist es, durch Prototyping eventuelle hohe Kosten zu vermeiden. In dem Arbeitspaket wird unter anderem von ein Prototyp erwähnt, der folgendes Ziel hat: „Der zu entwickelnde Prototyp zeigt Vorteile und Schwachstellen der spezifizierten Konzepte auf und dient so einer ersten Machbarkeitsstudie, welche die weitere Entwicklung maßgeblich bestimmt.". Für den Prototyp sollen zunächst „Ziele definiert und hieraus die zu entwickelnden Schnittstellen und Module definiert" werden, dabei findet die Zieldefinition in diesem Kapitel statt. Die Schnittstellen sind gegebenenfalls auch für das Arbeitspaket 2.200 „Systembeschreibung/-architektur" relevant.

Folgende Punkte stellen einen Auszug der Anforderungen an diese Diplomarbeit dar:

- Externe Literatur soll nach ähnlichen Problemstellungen und etwaigen Lösungen sowie sonstigen nützlichen Informationen zur Generierung von Benutzerschnittstellen durchsucht werden.
- Bisherige studentische Arbeiten am VIS sollen bei Bedarf zu Rate gezogen werden.
- Anforderungen an System und Schnittstellen werden auf Grundlage möglicher Test- und Vorführfälle erhoben.
- Da sich das Projekt IP-KOM-ÖV in einem frühen Stadium befindet und die Schnittstellen auf dieser Seite noch nicht festgeschrieben sind, soll eine möglichst eingängige Schnittstelle zum Ansteuern und Konfigurieren des Generators entworfen werden. Dies kann beispielsweise ein XML-basiertes Format oder eine domänenspezifische Konfigurationssprache sein, die einer späteren Anbindung der Testanwendungen an standardisierte Schnittstellen wie WebServices oder Ähnliches nicht im Weg steht.

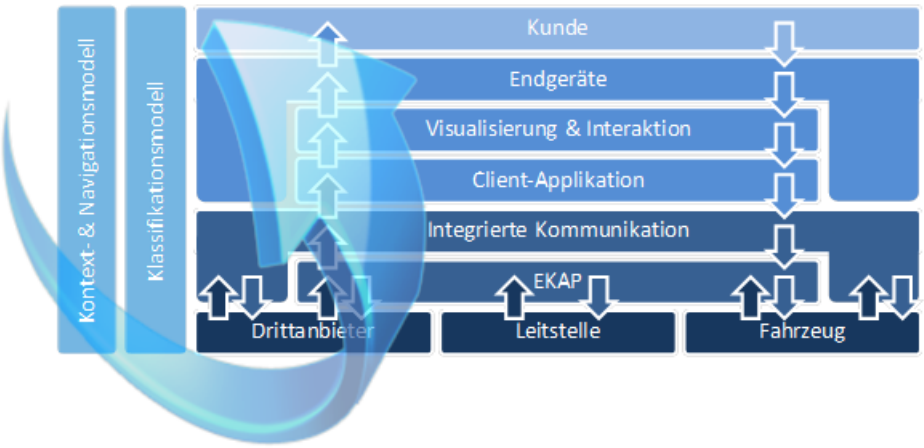


**Abbildung 2.1.:** Die geplante Architektur des IP-KOM-ÖV Projektes, die auch im entgültigen Prototyp umgesetzt werden soll.

- Der Generator soll so entwickelt werden, dass er auf den IP-KOM-ÖV-Entwicklungsrechnern ausführbar ist und Anwendungen erzeugt, die auf einem Mobilgerät ausgeführt beziehungsweise angezeigt werden können.
- Für die Zukunft ist auch zu überlegen, wie die GUI-Generierung direkt auf dem Mobilgerät stattfinden könnte.
- In der schriftlichen Ausarbeitung sollen alle Ergebnisse der Recherche, Analyse und der gesamte Entwicklungsverlauf beschrieben werden. Dabei soll auch aufgezeigt werden, inwieweit der hier entwickelte Generator mitsamt seiner Konfiguration, existierenden GUI-Generatoren bei der Entwicklung von IP-KOM-ÖV überlegen ist.

Die Testfälle wurden im Laufe der Arbeit festgelegt und befinden sich im Anhang A. Weitere Schnittstellen die teilweise in Zusammenarbeit mit dem Betreuer erstellt wurden, werden in der Konzeption beschrieben. Für die Wahl der mobilen Plattform wurde ebenfalls im Rahmes eines Gespräches mit dem Betreuer in der Anfangsphase folgende Anforderungen festgelegt:

- Es reicht das die Prototypen auf einer Plattform laufen. Eine plattformübergreifende Lösung ist nicht notwendig, kann aber verwendet werden, falls dies insgesamt die beste Lösung ist.
- Die Plattform sollte einen relevanten Marktanteil besitzen, so dass die Weiterentwicklung gesichert und ausreichend Dokumentation im Web vorhanden ist.

- Der Bildschirm sollte zeitgemäß sein, also ausreichend Farben darstellen sowie 2D Grafiken darstellen können. Eine Darstellung von 3D Grafiken wird dagegen nicht benötigt.
- Was die Rechenleistung betrifft, sollte die Reaktionszeit möglichst gering sein. XML Verarbeitung sollte performant unterstützt werden.
- Als Eingabe soll ein Touchscreen mit virtueller Tastatur dienen. Alternativ oder zusätzlich kann die Eingabe auch übere eine Hardware-Tastatur möglich sein.
- Optional soll es möglich sein Standardortinformationen zu bestimmen. Dies kann alternativ aber auch durch einen Stub ersetzt werden.
- Neben den anderen genannten Kriterien ist auch die Präferenz des Diplomanten entscheidend.

Die Ergebnisse dieser Arbeit sollen im Laufe des Projekt noch erweitert werden, voraussichtlich über die nächsten zwei bis drei Jahren. Daher ist insbesondere eine gute Entwicklerdokumentation wichtig, den optimalerweise kann der bestehende Prototyp einfach weiterentwickelt werden.

Weiterhin sollen auch die Erkenntnisse, die im Zusammenhang mit der modellgetriebenden Softwarentwicklung in der Arbeit gewonnen werden, gut dokumentiert werden.

# 3. Grundlagen

In diesem Kapitel werden verfügbare Konzepte und Technologien vorgestellt und evaluiert. Im Abschnitt 3.1 werden mobile Plattformen für Smartphones vorgestellt und auf die Tauglichkeit für den Einsatz im Projekt untersucht. Im Abschnitt 3.2 werden die Konzepte und die Anwendung der modellgetriebenen Softwareentwicklung erläutert. In Abschnitt 3.4 werden verfügbare Verkehrsinformationssysteme, insbesondere für mobile Plattformen, untersucht.

## 3.1. Mobile Plattformen

Mobile Geräte haben sich in den letzten Jahren von einfachen Handys, über Telefone mit Zusatzfunktionen, so genannte Feature Phones [Fea], zu vollwertigen Plattformen mit einer großen Auswahl an Anwendungen entwickelt. Diese werden allgemein als Smartphones bezeichnet [Sma]. Für IP-KOM-ÖV ist die Auswahl der Plattform insbesondere deshalb von großer Bedeutung, da das Projekt erst in mehreren Jahren abgeschlossen sein wird und sich bis dahin noch einiges auf dem Markt der mobilen Plattformen ändern kann. So wurde beispielsweise Windows Mobile 6.5 letztes Jahr obsolet [Win10], um ein weiteres Sinken der Marktanteile zu vermeiden [Sma11]. Android konnte dagegen als Spätstarter seinen Markanteil relativ zügig ausbauen, wie Abbildung 3.1 zeigt.

Seit der Einführung des iPhones von Apple hat sich die Smartphone-Landschaft grundlegend gewandelt. Davor waren Smartphones ein Nischenprodukt, das hauptsächlich Geschäftskunden bediente. Heute entwickeln sich Smartphones mehr und mehr zum Massenprodukt. Diesen Erfolg hat Apple insbesondere seiner und der Software von Dritt-Herstellern zu verdanken. Über die App Store können Dritt-Hersteller eigene Programme anbieten, auf die jeder Nutzer der iOS[1]-Plattform zugreifen kann. Konkurrenten wie Google und Microsoft folgten diesem Konzept und haben eigene Marktplätze für Smartphone-Apps geschaffen. Während bei einfachen Handys und sogenannten Feature-Phones die Software und Bedienung noch eine untergeordnete Rolle spielen, rücken diese beiden Aspekte bei Smartphones in den Fokus.

Auch Apps für Verkehrsinformationssysteme gibt es schon in großer Zahl. So gibt es zum Beispiel allein im Android Market über 1000 kostenlose und kostenpflichtige Apps in der Kategorie Verkehr [Andd] und die offizielle App der Deutschen Bahn wurde bereits eine Million Mal heruntergeladen [DBN]. Die meisten Anwendungen sind dabei zur Zeit noch

[1]Name des Betriebssystems des iPhones

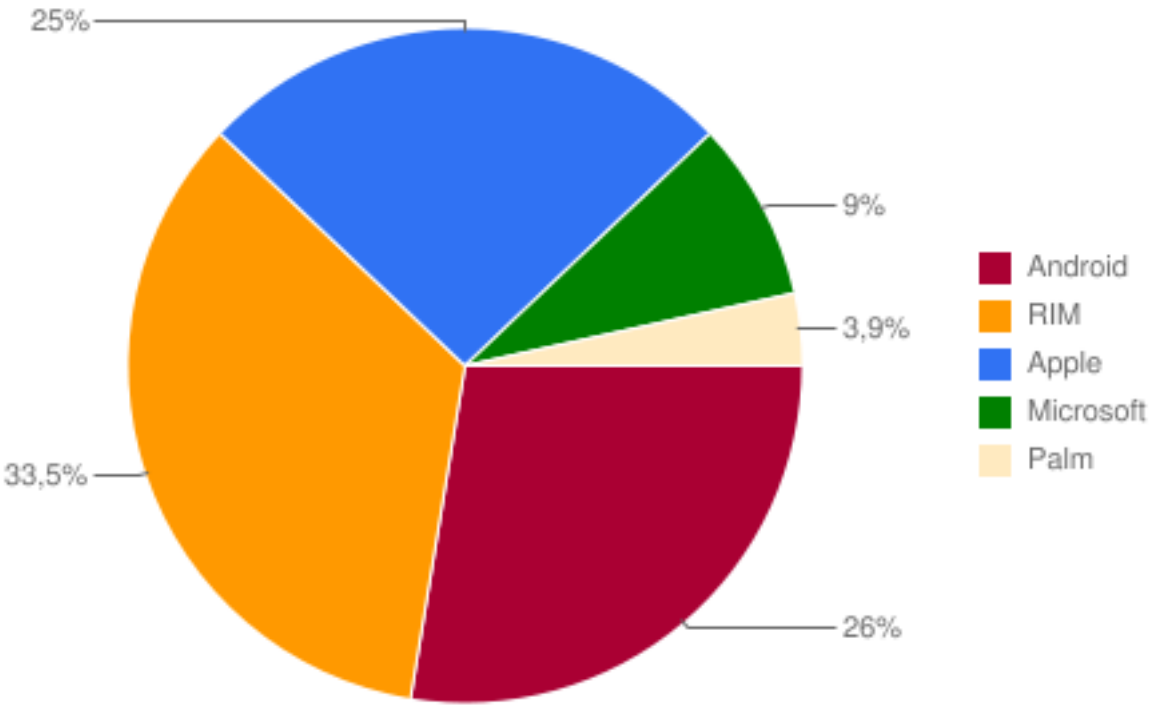


**Abbildung 3.1.:** Anteile der Betriebsysteme der aktuelle gekauften Smartphones in der USA laut Umfrage von comScore im Februar 2011 [Sma11]

relativ einfach gestaltet und besitzen oft nur einen Bruchteil der Funktionalität, die zum Beispiel über die Webanwendungen der jeweiligen Verkehrsunternehmens möglich ist. Auch gibt es zur Zeit noch wenig Integration mit anderen Diensten, wie zum Beispiel mit sozialen Netwerken oder Location-based Services.

Dieser Abschnitt soll einen kurzen Überblick über die verschiedenen mobilen Plattformen geben und anschließend anhand der Kriterien der Aufgabenstellung beurteilen, inwieweit diese für Prototypen für mobile Plattformen des Projektes geeignet sind.

### 3.1.1. Apple iOS

Mit iOS bezeichnet Apple die Plattform für ihr Smartphone iPhone sowie für ihr Tablet iPad. In dieser Arbeit wird nur ersteres betrachtet, wobei die Anwendungsentwicklung für die beiden Geräteklassen weitgehend indentisch erfolgt. Auf dem iPad sind allerdings wegen des größeren Bildschirmes und der höheren Auflösung und des dadurch bedingten zusätzlichen Platzes auch mehrere Panels in einer Anwendung möglich. Zum Beispiel verfügt die Mail App in der iPad Version über ein Panel zum Auswählen von Mails und ein Panel zum Anzeigen der ausgewählten Mail.

Die aktuellste Version von iOS ist 4.3. Die Entwicklungssprache und -werkzeuge entsprechen im wesentlichen denen von Mac OS X, die Entwicklung ist daher auch nur auf einem Mac möglich. Der C-Dialekt Objective-C [Obj] dient als Programmiersprache und die mobile Variante des Framework Cocoa [Coc] wird für die Oberfläche verwendet. Eigene Anwendungen lassen sich nur auf dem Gerät installieren, wenn man dem Entwicklerprogramm von Apple für eine Jahresgebühr von hierzulande 99 Euro beitritt[2] [iOSa]. Anwendungen können zudem nur über die App Store verteilt werden, die allerdings auch auf eine bestimmte Benutzergruppe beschränkt werden kann. Öffentliche Apps werden einem Reviewprozess unterzogen, der neben der Funktionsfähigkeit der App unter anderem auch prüft, ob die Richtlinien zur Gestaltung der Anwendung von Apple eingehalten wurden [iOSb].

### 3.1.2. Android

Android [Andb] wird im Gegensatz zu den meisten anderen Smartphone-Betriebssystemen nicht von einem einzelnen Hersteller entwickelt, sondern durch das Konsortium Open Handset Alliance [OHA], das von Google geführt wird. Die Quellen von Android sind größtenteils frei verfügbar. Es wird daher von Herstellern nicht nur auf Smartphones eingesetzt, sondern auch auf anderen Geräteklassen wie beispielsweise Netbooks und Tablets. Die Version 3.0 enthält auch offiziell Tabletunterstützung, mit der Anwendungen auf Tablets den zusätzlich zur Verfügung stehenden Platz besser nutzen können.

Android liegt aktuell für Smartphones in der Version 2.3 vor. Die Entwicklung ist plattformunabhängig mittels Java möglich. Google hat hierfür mit der Dalvik Virtual Machine eine eigene Java Virtual Machine entwickelt [Dal]. Diese ist so ressourcenschonend, dass problemlos mehrere Instanzen auf einem Gerät laufen können. Daher verfügt Android auch von Anfang an über echtes Multitasking. Ebenso ist es möglich C++ zu verwenden, was dann aber nicht mehr plattformunabhängig ist. Entwicklungswerkzeuge werden für alle gängigen Betriebssysteme angeboten. Bezüglich der Weitergabe der Anwendungen gibt es keine Einschränkungen.

### 3.1.3. RIM BlackBerry

BlackBerry OS [Blab] ist vor allem in der Geschäftswelt weit verbreitet. Es wird nur von der kanadischen Firma RIM auf gleichnamigen Geräten verwendet. BlackBerrys verfügen meist über eine Hardware-Tastatur, wobei mittlerweile immer mehr Geräte entweder mit zusätzlichem oder gar außschließlich mit Touchscreen angeboten werden. Eine weitere Stärke von Blackberrys ist ihr Push-Service, mit dem E-Mails überall dort empfangen werden können, wo Netzabdeckung besteht. Dieser Service kostet allerdings meistens extra bei den Netzanbietern und verliert durch die immer bessere Verfügbarkeit des mobilen Internets daher an Bedeutung. Die Entwicklung dafür ist mit C/C++ und Java SDK möglich [Blaa], auch Java ME wird unterstützt. Mit dem PlayBook bietet auch RIM ein Tablet an [Pla].

[2] Stand 2011

### 3.1.4. Windows Phone 7

Windows Phone 7 ist eine komplette Neuentwicklung von Microsoft, nachdem ihr vorheriges Smartphone-Betriebssystem Windows Mobile 6.5 zunehmend an Marktanteilen verloren hat. Es ist daher auch inkompatibel zu früheren Windows Smartphone-Betriebssystemen. Von den Features und Beschränkungen lehnt es sich stark an iOS an. Bei der Oberfläche versucht es aber neue Wege zu gehen. Der Startbildschirm besteht aus sogenannten Kacheln, die kleine Informationshäppchen, wie die Anzahl der ungelesenen E-Mails oder den aktuellen Facebook-Status, anzeigen. Als Entwicklungssprache wird C# mit dem Silverlight Framework verwendet. Alternativ kann Visual Basic .NET als Entwicklungssprache eingesetzt werden [VBN]. Das SDK ist, wie alle Entwicklungswerkzeuge von Microsoft, nur unter Windows verfügbar.

### 3.1.5. Palm webOS

Die Firma Palm ist vor allem durch ihre Erfolge im Bereich PDAs (Personal Digital Assistant) bekannt. Nachdem ihr Smartphone-Betriebssystem Palm OS mit dem iPhone nicht mehr mithalten konnte, wurde Anfang 2009 dessen Nachfolger webOS vorgestellt [web]. Es basiert auf Linux und ist wie iOS und Android auf die Bedienung per Touchscreen angepasst. Entwickelt werden kann auf webOS, passend zum Namen, mit den für Webanwendungen zur Verfügung stehenden Hilfsmitteln: HTML 5, CSS und JavaScript. Seit März 2010 ist es auch möglich C- und C++-Code in Anwendungen zu verwenden.

Smartphones mit webOS sind ausschließlich von Palm erhältlich. Die Firma Palm wurde im April 2010 von HP übernommen und firmiert seit Oktober 2010 als HP Palm. HP hat für Juni 2011 auch Tablets mit webOS angekündigt [HPP].

### 3.1.6. Weitere Smartphone-Betriebssysteme

Neben den Smartphone Plattformen die sich zur Zeit gut am Markt verkaufen [Sma11] gibt es noch ein paar weitere erwähnenswerte Smartphonebetriebssysteme.

Nokia pflegt gleich zwei Smartphonebetriebsysteme. Zum einen das relativ alte Symbian [Sym], das vor allem im Low-Cost-Smartphone Markt zufinden ist. Zum anderen MeeGo [Mee], das aus Maemo [Mae] und Moblin [Mob] hervorgegangen ist und im höherpreisige Marktsegmente angesidelt ist. Für beide Systeme kann man mit dem C++ Framework Qt entwickeln. Symbian unterstützt außerdem Java ME.

Auch Samsung hat 2010 mit Bada [Bad] ein eigenes Smartphone-Betriebssystem veröffentlicht. Es zielt vor allem auf den Low-Cost-Smartphone Markt ab [Nor10]. Entwickelt wird mit einem eigenen C++ SDK. Java ME wird ebenfalls unterstützt.

### 3.1.7. Fazit

Hauptkriterien für die Auswahl der mobilen Plattform für diese Arbeit sind niedrige Investitionskosten, eine gewissen Zukunftssicherheit sowie eine brachbare Entwicklungsumgebung.

Die größten Investitionskosten fallen bei der iOS Plattform an. Neben einem iPhone, benötigt man für die iOS-Entwicklung nämlich auch zwingend einen Mac von Apple. Gegen Windows Phone 7 spricht dagegen hauptsächlich die noch recht geringe Verbreitung und die Frage ob sich die relativ spät gestartete Plattform noch gegenüber der Konkurrenz behaupten kann. Gleiches gilt auch für webOS. Auch für BlackBerry ist die Zukunft offen, da man zwar stark im Geschäft mit Unternehmen ist, im Konsumermarkt kaum Fuss fassen kann. Dies dürfte vor allem daran liegen, das die meisten Modelle immer noch keinen Touchscreen besitzen. Die Wahl fällt daher auf Android, das alle Kriterien ohne Einschränkungen erfüllt. Android-Geräte gibt es teilweise schon für unter 100 Euro [Che] und die Entwicklungswerkzeuge sind kostenlos und laufen auf allen relevanten Betriebssystemen. Es gibt eine große Zahl an Geräteherstellern, die Plattform wird durch eon Konsortium wntwickelt und ist weitgehend frei verfügbar. Man besitzt zudem bereits eine gewissen Marktanteil [Sma11]. Als Entwicklungssprachen kommen die Top 3 der verbreitetsten Sprachen zum Einsatz [TIO].

## 3.2. Modellgetriebene Softwareentwicklung

In diesem Abschnitt werden die Grundlagen der modellgetriebenen Softwareentwicklung vorgestellt. Im Abschnitt 3.2.1 wird der Begriff der Modelle genauer erläutert. In 3.2.2 wird die modellgetriebene Softwareentwicklung definiert. Schließlich wird in 3.2.3 auf domainspezifische Sprachen eingegangen, die für den Einsatz von modellgetriebener Softwareentwicklung notwendig sind. In den letzten beiden Abschnitten 3.3.1 und 3.3.2 werden dann konkret Sprachen und Werkzeuge für domainspezifische Sprachen vorgestellt und verglichen.

### 3.2.1. Modelle

In diesem Abschnitt wird die Bedeutung von Modellen kurz zusammengfasst. Die Erklärungen dazu wurde dabei weitgehend aus [JL06] entnommen.

Modelle treten in den verschiedensten Bereichen auf und sind ein wichtiges Hilfsmittel unserer Welt. Beispielsweise sind Fotos, Fingerabdruck oder Matrikelnummer Modelle. Es gibt zwei Arten von Modellen, deskriptive (d.h. beschreibende) und präskriptive (d.h. vorschreibende) Modelle. Ein Foto ist ein Abbild, als deskriptiv. Ein Bauplan ist dagegen eine Vorgabe, also präskriptiv. Im Informatikbereich sind alle Artefakte Modelle. Ob Spezifikation, Entwurf oder Code, alles sind Modelle, die dazu eingesetzt werden, einen bestimmten Ausschnitt der Softwareentwicklung abzubilden. Diese Modelle werden ineinander überführt. Anhand des Entwurfs wird der Code geschrieben. Dabei sind nicht alle Details des Entwurfs im Code berücksichtigt und umgekehrt. Die präterierten Attribute (vom lat. praeter: außer,

ausgenommen) fallen weg und es kommen die abundaten Attribute (vom lat. abundans: übervoll, überreich) hinzu, die nichts mit dem Orginal zu tun haben. Beide Eigenschaften sind für die modellgetriebene Entwicklung wichtig. Denn dort wird in der Regel ein einfaches Modell in ein reichhaltigeres Modell überführt, wobei aber nicht unbedingt immer das ganze Modell berücksichtigt wird. Zum Beispiel kann es Teile geben, die nur für bestimmte Varianten bestimmt sind oder integrierte Modelldokumentation, die im anderen Modell wegfällt.

### 3.2.2. Definition Modellgetriebene Softwareentwicklung

Zum Thema modellgetriebene Softwareentwicklung findet man in der Literatur verschiedene Definitionen. Das Buch „Modellgetriebene Softwareentwicklung“ [TS07] enthält folgende Definition:

„Modellgetriebene Softwareentwicklung (Model Driven Software Development, MDSD) ist ein Oberbegriff der Techniken, die aus formalen Modellen automatisiert lauffähige Software erzeugen.“

Es muss also erst einmal ein formales Modell vorliegen. Formal heißt, dass es klare Regeln gibt, die über das Modell Aussagen macht. Dies ist notwendig, damit der Prozess automatisiert ablaufen kann. Denn ein nicht formales Modell kann ein Rechner nur schwer verarbeiten. Zudem soll der Prozess automatisiert sein, das heißt vereinfacht gesagt, das man einfach ein Modell eingeben kann und dann per Knopfdruck das Zielartefakt daraus erzeugt. Der dritte Teil der Definition spricht von lauffähiger Software, es soll also nicht noch zusätzliche Handarbeit notwenig sein, um das Programm lauffähig zu machen.

### 3.2.3. Domänenspezifische Sprachen

Für modellgetriebene Entwicklung benötigt man eine Abstraktionssprache, die dann in eine oder mehrere Zielsprachen überführt wird. Dies sind die sogenannten domänenspezifischen Sprachen, abgekürzt mit DSL[3]. Eine DSL besteht üblicherweise aus zwei Teilen: Der Sprache, also der Syntax, die die Domäne beschreibt und einem Codegenerator, der aus der Eingabe das entsprechende Endprodukt erzeugt. Dies wird oft über Templateengines, wie zum Beispiel StringTemplate [Str], realisiert. Eine alternative Möglichkeit wäre die DSL direkt zu interpretieren. Dies ist aber nicht in allen Fällen, zum Beispiel wenn die Zielplattform eine andere ist als die Entwicklungsplattform ist, möglich.

[3]Domain specific language (englische Bezeichung)

#### Textuelle vs. grafische DSLs

Eine DSL kann sowohl textuell als auch graphisch dargestellt werden. Eine bekannte graphische DSL ist zum Beispiel UML. Graphische Darstellungen sind zwar für den Menschen meist leichter verständlich, allerdings gilt dies meist nur für einfache Modelle. Ein weiterer Nachteil ist, dass ein spezielles Programm zur Anzeige benötigt wird. Auch für Operationen für die man bei Text auf Standardwerkzeuge zurückgreifen kann, wie zum Beispiel ein Diff, benötigen Werkzeuge, die dies unterstützen. Eine bekanntes Werkzeug zum Erstellen von grafischen DSL ist zum Beispiel das kommerzielle Produkt MetaEdit+ [Met]. Diese Diplomarbeit beschränkt sich allerdings auf Werkzeuge zur Erzeugung von textuellen DSLs. Eine Auswahl wird im nächsten Abschnitt vorgestellt.

#### Interne DSL vs. Externe DSL

Als interne DSL bezeichnet man eine in eine bestehende Sprache eingebettete DSL. Diese benutzt nur eine Teilmenge der Syntaxelemente der Hostsprache und lässt sich so im Idealfall auch von jemandem verstehen, der die Hostsprache nicht beherrscht. Typischerweise sind dynamische Sprachen dafür besser geeignet, es gibt aber mittlerweile auch statische Sprachen, die gut für die Implementierung von internen DSLs geeignet sind. In dieser Arbeit werden Ruby und Scala näher betrachtet. Prinzipiell sind aber auch Sprachen wie Python oder C# geeignet.

Eine externe DSL ist dagegen eine eigene Sprache, die meist mit Hilfe eines Parsergenerators entsteht. Sie ist dadurch zwar viel flexibler, aber auch aufwendiger zu erstellen. Denn für eine eigene Sprache gibt es erst einmal keine Werkzeugunterstützung. Diese zu erstellen kann unter Umständen genauso viel Zeit in Anspruch nehmen, wie die Erstellung der Sprache selbst. Die Lösung dieses Problems sind sogenannte Language Workbenches [Fow05]. Diese abstrahieren typischerweise den Parsergenerator, stellen ein Templatesystem zur Verfügung und generieren auch eine passende Umgebung zum Bearbeiten der fertigen DSL. In dieser Arbeit werden Eclipse Xtext und Jetbrains MPS als Vertreter von Language Workbenches betrachtet. Ein weiteres bekannteres Werkzeug ist der „Workbench“ von Intentional Software [Intb].

## 3.3. Besehende Ansätze zur Generierung von mobiler Oberflächen

Vergleiche zu Werkzeugen und Frameworks zur Generierung von grafischen Oberflächen werden asuführlich in [DKW10] behandelt. Für mobile Oberflächen gibt es zur Zeit noch wenig Werkzeuge. Ein Werkzeug für die Entwicklung für iPhone Apps ist iphonical [iph]. Bei ihm definiert man mit Hilfe einer eigenen DSL Domainklassen und kann daraus dann entsprechende Formulare generieren, mit denen man Entities auflisten anlegen, ändern und löschen kann. Das Projekt Applause [app] unterstützt neben dem iPhone auch noch das iPad und Android. Es vereinfacht die Entwicklung der Oberfläche durch eine DSL, die die graphischen Komponenten abstrahiert. beide Werkzeuge basieren auf Xtext (siehte 3.3.2).

### 3.3.1. Sprachen für DSLs

Dieser Abschnitt betrachtet eine Auswahl an Sprachen und Werkzeugen für DSLs. Bei der Auswahl wurde dabei praktische Einsatzfähigkeit und Verbreitung berücksichtigt.

Im Rahmen der Evaluation wurde versucht, mit den jeweiligen Sprachen und Werkzeugen die Pseudosyntax 3.1 abzubilden.

```
travel {
    group {
        start,
        destination
    },
    startdate,
    transportation("Train", "Subway")
}
```

**Listing 3.1:** Pseudo-DSL für die Evaluation von Werkzeugen und Sprachen für DSLs

Das Schlüsselwort "travel" leitet den Hauptteil der DSL ein. Mit Hilfe des "group" Schlüsselwortes können beliebige verschachtelte virtuelle Gruppen gebildet werden. "start", "destination", "startdate" und "transporation" beschreiben die Elemente, die angezeigt werden sollen. "transportation" hat dabei zusätzlich noch Parameter, die die möglichen Verkehrsmittel beschreiben.

#### Ruby

Die Programmiersprache Ruby [Ruba] wurde 1995 von Yukihiro Matsumoto entworfen. Sie wurde durch das Webframework „Ruby on Rails" [Rubb] populär. Ruby on Rails wird oft auch als DSL bezeichnet [Rub08]. Ruby ist dynamisch typisiert, das heißt Typen müssen nicht deklariert werden, sondern werden zur Laufzeit ermittelt. Die Sprache bietet außerdem umfangreiche Möglichkeiten zur Laufzeit Code zu generieren. Dies wird auch als Metaprogrammierung bezeichnet [Per10]. Die Eigenschaft, dass Ruby bereits einen Codegenerator eingebaut hat, begünstigt die Erstellung von DSLs.

```
travel {
    group {
        feature :start, :destination
    }
    feature :startdate
    transportation ["Train", "Subway"]
}
```

**Listing 3.2:** Die Pseudo-DSL in Ruby

In dem Beispiel 3.2 werden insbesondere Closures [Fow04] sowie die Eigenschaft, dass man Klammern bei der Übergabe von Parametern oft weglassen kann benutzt. Closures sind vereinfacht gesagt anonyme Funktionen.

```
def hello
  print 'Hello '
  # Hier wird der zu \"ubergebende Codeblock eingesetzt
  yield
  print '!'
end
hello { print 'World' }
```

**Listing 3.3:** Einfaches Beispiel für eine Closure in Ruby. Ausgabe: Hello World!

#### Scala

Scala entstand Anfang dieses Jahrtausends an der Schweizer Hochschule EPFL. Während bei der ersten Version noch stark der Forschungsaspekt im Vordergrund stand, ist es mittlerweile auch ein Ziel, Scala in der Industrie zu etablieren. Scala läuft auf der Java Virtual Machine [LY99] und ist statisch typisiert, das heißt der Typ einer Variable muss prinzipiell immer dem Compiler bekannt sein. Im nächsten Abschnitt werden einige Methoden erläutert, die sicherstellen, dass Scala trotzdem noch genügend flexibel ist für den Bau von DSLs.

In dem Beispiel 3.4 werden Type-Inferenz sowie die Apply Methoden benutzt. Type-Inferenz bedeutet, dass der Datentyp vom Compiler automatisch erkannt wird und nicht bei jeder Deklaration explizit angegeben werden muss. Die Apply Methode ist eine vereinfachte Darstellung, die meist für das Factory-Pattern benutzt wird. In Scala kann jede Klasse ein sogenanntes Kompanien-Object haben, von dem aus auch Interna der Klasse zugegriffen werden kann. Dieses Object kann eine Apply Methode besitzen bei der man statt Object.apply(Parameter) auch verkürzt Object(Parameter) schreiben kann. Weitere nützliche Eigenschaften für DSLs sind die Infix-Operator-Notation und implizite Konvertierung von Typen.

```
val travel = List[Feature](
    Group(Feature(start, destination)),
    Feature(startdate),
    Transportation("Train", "Subway")
)
```

**Listing 3.4:** Die Pseudo-DSL in Scala

### 3.3.2. Werkzeuge für DSLs

Wie in Abschnitt 3.2.3 schon erwähnt, gibt es neben internen DSLs, auch externe DSLs. Hierzu gibt es spezielle Werkzeuge, die auch als Lanuage Workbenches [Fow05] bezeichnet werden.

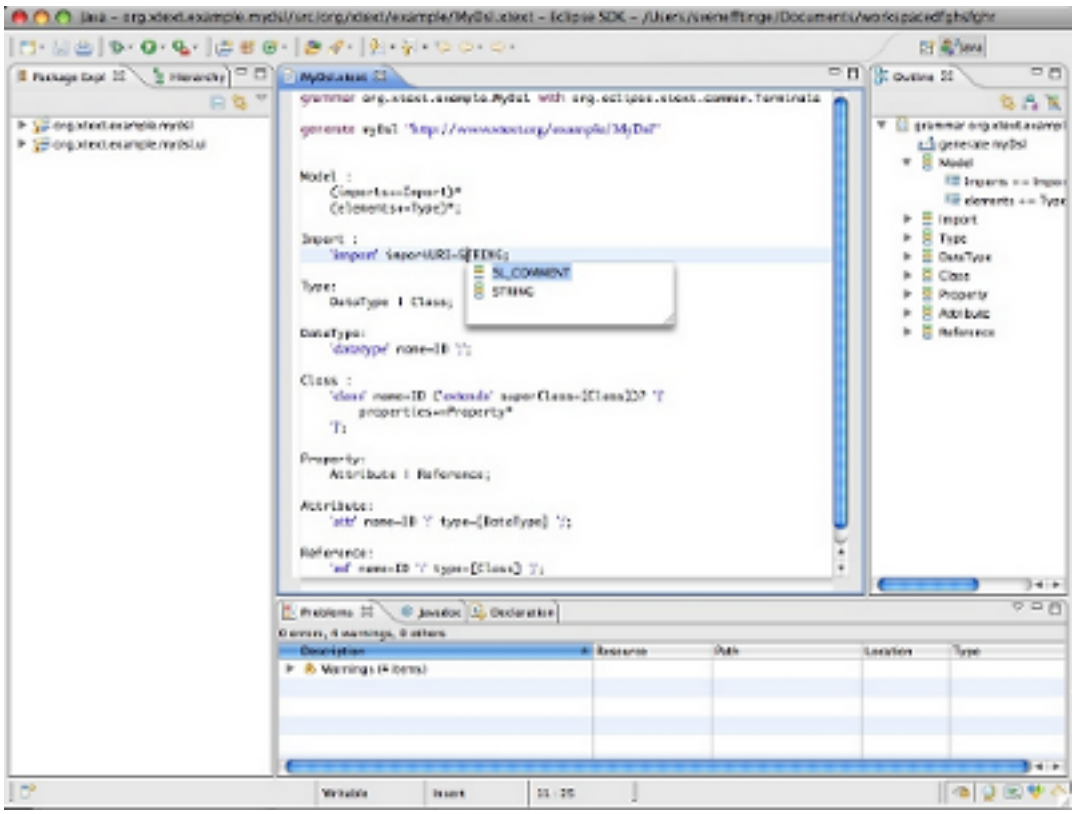

**Abbildung 3.2.:** Das Hauptfenster von Xtext

### Xtext

Xtext [Xteb] (siehe Abbildung 3.2) ist ein Open-Source-Framework für die Entwicklung domainspezifischer Sprachen. Es entstand aus dem openArchitectureWare-Projekt und ist mittlerweile Teil des Eclipse Modeling Projektes. Aus der von Xtext definierten textuellen Syntax wird sowohl eine ANTLR [ANTb] Grammatik als auch ein Editor für Eclipse [Ecl] erzeugt. Im Hintergrund steht zudem ein generiertes EMF Modell [EMF], das entweder direkt interpretiert werden kann oder zum Beispiel mittels einer Templatesprache weiter transformiert werden kann.

Xtext wird maßgeblich von der deutschen Firma itemis entwickelt [Xted], die dazu Beratung und Entwicklung anbietet. Es wird auch in der Industrie eingesetzt [Xtea].

### Jetbrains MPS

Das von der Firma Jetbrains entwickelte MPS (siehe Abbildung 3.3) steht für "Meta Programming System". Der Hauptanwendungsbereich ist das Erweitern von Sprachen. Zum

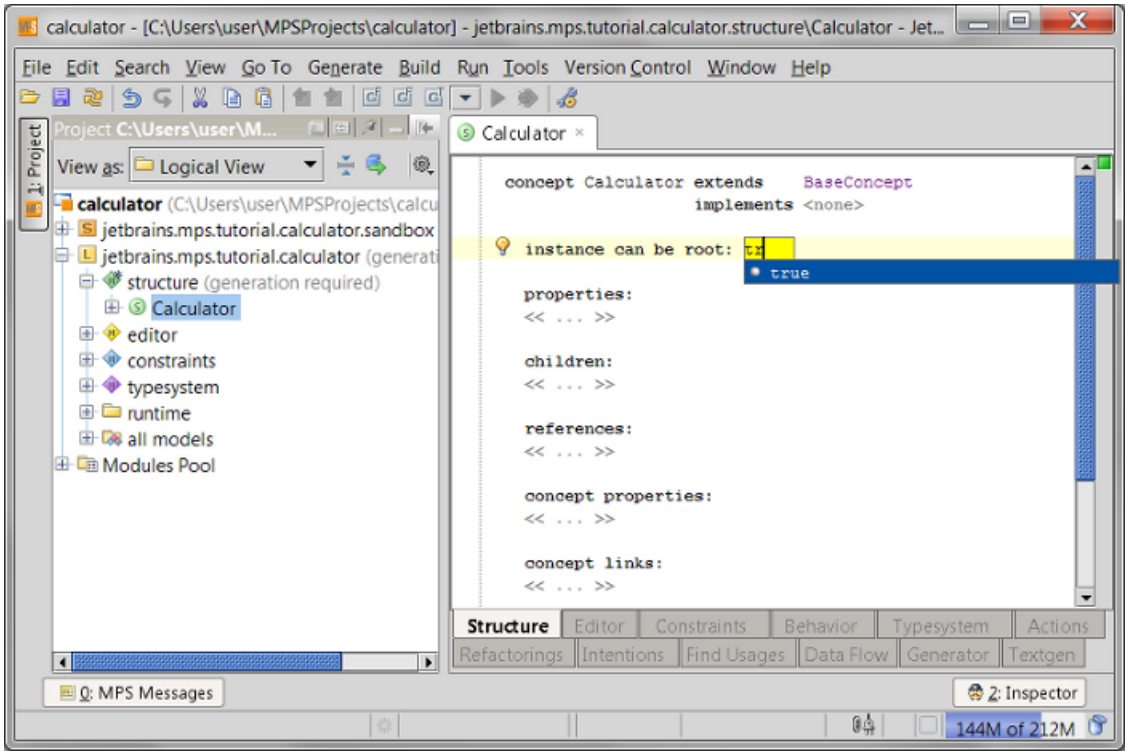


**Abbildung 3.3.:** Das Hauptfenster von Jetbrains MPS

Beispiel könnte Java mittels MPS um Konstrukte zur parallelen Programmierung erweitert werden. Eine weitere Besonderheit ist, dass dies nicht auf rein textueller Basis erfolgt, sondern mittels einer Meta-Ebene, die zwar textuell aussieht, aber die eigentliche Struktur versteckt (WYSIWYG). Wie Xtext erzeugt auch MPS einen Editor mit Komfortfunktionen wie Autovervollständigung und Outline-Funktion.

MPS ist Open Source und wurde von JetBrains unter anderem für ihr eigenes Produkt YouTrack eingsetzt. Ansonsten wird auch profesioneller Support angeboten [MPS]. Mit dem Realaxy ActionScript Editor gibt es auch die erste kommerzielle IDE, die auf MPS basiert [Rea].

### 3.3.3. Fazit

Alle Sprachen und Werkzeuge haben sich im Test als brauchbar erwiesen. Da um flexibel zu bleiben eine externe DSL erstellt werden soll, blieb die Wahl aber zwischen den zwei Werkzeugen für DSL. Die Wahl fiel auf Xtext, da die Einstiegshürde nieriger ist. Jetbrains MPS hat zudem seine Stärke eher beim erweitern bestehender Sprachen, was für diese Arbeit nicht in Betracht kommt.

## 3.4. Verkehrsinformationssysteme

Erst einmal gilt es zu klären, was genau überhaupt ein Verkehrsinformationssystem ist. Recherchen ergaben dazu ernüchternde Ergebnisse, der Begriff taucht zwar häufiger auf, aber eine einheitliche Definition scheint es nicht zu geben. In einer Diplomarbeit von 2000 am Institut für Straßen- und Verkehrswesen der Universität Stuttgart [Hoboo] zählen hierzu, neben Mobilfunk-, Internet-Diensten und WAP, unter anderem Verkehrsnachrichten im Radio, digital über Radiofrequenzen übertragene programmbegleitende Informationen und Videotext. In dieser Arbeit sind als Verkehrsinformationssysteme in der Regel solche gemeint die auf mobilen Plattformen, genauer Smartphones, laufen. Eine weitere Einschränkung ist, dass sich Verkehrsinformationssysteme in dieser Arbeit hauptsächlich auf das Verkehrsmittel Bahn konzentrieren. Weitere öffentliche Verkehrmittel wie Bus und Taxis sollen aber gegebenfalls eingebunden werden.

Es gibt bereits eine Auswahl an Verkehrsinformationssystemen für mobile Plattformen. Die meisten sind recht ähnlich aufgebaut. Es gibt eine Suchmaske für Verbindungen, die zu einer Ergebnisliste führt. In der Regel kann man durch Auswahl der gewünschten Verbindung dann noch einmal Details dazu abrufen.

In diesem Abschnitt werden ein paar Vertreter kurz beispielhaft vorgestellt.

### 3.4.1. DB Navigator

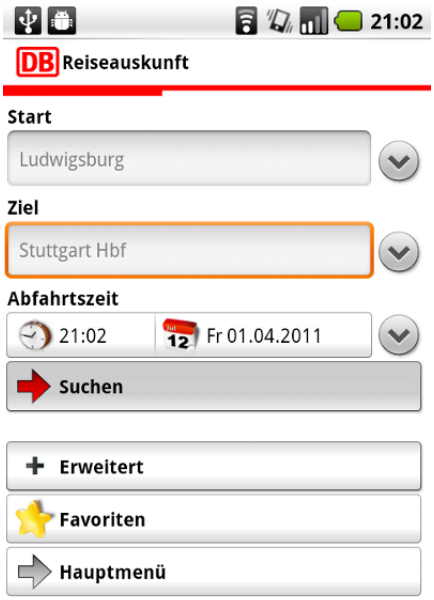


**Abbildung 3.4.:** Das Hauptfenster des DB Navigator

Die Deutsche Bahn bietet Apps für verschiedene Smartphone Betriebsysteme an, unter anderem iPhone, Android (siehe Abbildung 3.4), BlackBerry, Symbian und Bada [DBN]. Daneben gibt es auch eine Java ME Implementierung. Die Grundfunkionalität ist dabei das Suchen von Verbindungen aus der umfangreichen Datenbank der Bahn. Einzelne andere Features können abweichen. Auch sind nicht alle Funktionen verfügbar, die über das Web verfügbar sind. Die Funktion "'Ist mein Zug pünklich?' ist zum Beispiel nur auf der mobilen Webseite verfügbar, wird aber in den offiziellen Apps der Bahn nicht unterstützt.

### 3.4.2. VVS

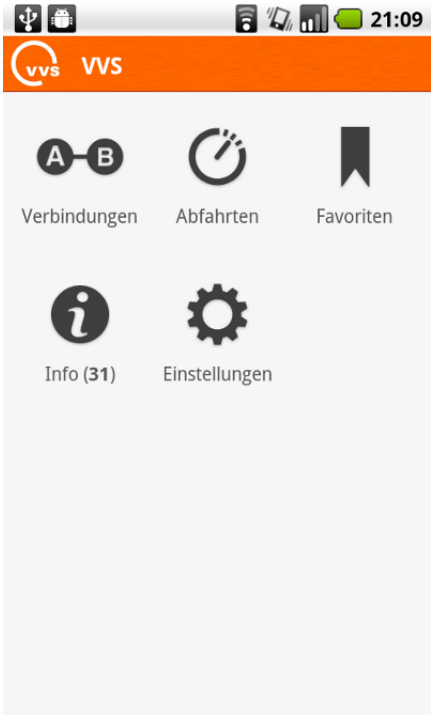


**Abbildung 3.5.:** Das Hauptfenster der VVS App

Der VVS (Verkehrs- und Tarifverbund Stuttgart) bietet seit Ende 2010 auch Apps für iPhone und Android (siehe Abbildung 3.5) an [VVS]. Die Oberfläche wirkt allerdings nicht ganz so durchdacht wie bei der Bahn App, so kann man einen Standort nicht einfach eintippen, sondern muss diesen immer erst suchen beziehungsweise direkt auswählen. Zudem werden in der Detailansicht auch nicht die Informationen zu den Abfahrts- und Ankuftsgleisen angezeigt. Gut ist dagegen, dass vorliegende Störungen direkt als Popup angezeigt werden.

### 3.4.3. Öffi


**Abbildung 3.6.:** Das Hauptfenster von Öffi

Öffi [Oef] ist eine kostenlose App für Android (siehe Abbildung 3.6), die privat entwickelt wird. Sie enthält viele Pläne von Regionen in Deutschland, sowie einige wenige von anderen Regionen in Europa, Australien und Amerika. Das Projekt besteht genauer gesagt aus zwei Apps: Eine App die nach Haltestellen in der Nähe sucht und dafür die nächsten Abfahrtzeiten anzeigt und eine App für Verbindungsuche. Die Einstellungsmöglichkeiten sind dabei eher minimal, dafür kann die App verfügbare Verbindungen visualisieren. Sie kann auch komplette Netzpläne anzeigen. Weiterhin ist die App sehr gut in Android integriert. Man kann Öffi zum Beispiel direkt von der Adresse eines Kontakts im Adressbuch aufrufen.

### 3.4.4. push&ride

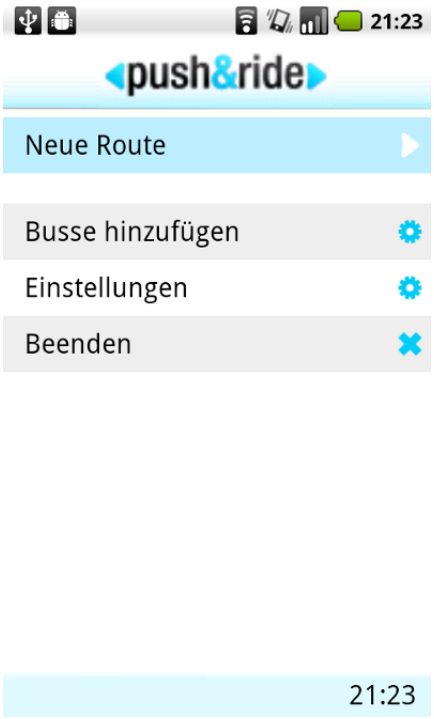


**Abbildung 3.7.:** Das Hauptfenster von Push & Ride

push&ride [Pus] (siehe Abbildung 3.7) ist eine Java ME Anwendung und funktioniert im Gegensatz zu den anderen Apps komplett offline. Sie hat dafür allerdings auch nur den VVS Plan der Region Stuttgart. Zudem ist es keine Smartphone App im eigentlichen Sinne, es gibt zwar auch eine Android Portierung, die allerdings weitgehend aus dem eingebetteten Java ME Programm besteht. So unterstützt die Android App nicht die virtuelle Android Tastatur, sondern man muss Eingaben umständlich über eine eigene abgespeckte Tastatur eingeben. Außerdem werden auch teilweise Bildschirmelemente abgeschnitten. Dafür unterstützt push&ride aber auch viele Feature Phones.

### 3.4.5. Fazit

Zusammengefasst unterscheiden sich bestehende Verkehrsinformationssysteme für mobile Geräte bisher nicht besonders voneinander. Auch im Vergleich zu Verkehrsauskünften über Webseiten oder stationäre Automaten sind keine allzu großen Unterschiede zu erkennen. Zwar gibt es durchaus hilfreiche Funktionen wie Favoriten und GPS-Ortung, ansonsten fehlen innovative Funktionen bisher. Auch wird in keiner Anwendung zwischen verschiedenen Benutzertypen unterschieden.

Am besten schlägt sich der DB Navigator der Deutschen Bahn. Neben der Tatsache, dass die Anwendung eine sehr guten Datenbasis besitzt, ist die Anwendung von der Oberfläche, Bedienung und Performance, zumindest in der Android-Version, am besten. Aber auch alle anderen Anwendungen bieten trotz mancher Unzulänglichkeiten, Alleinstellungsmerkmale die für ihren Einsatz sprechen.

# 4. Konzeption

Dieses Kapitel beschreibt die Konzeption der DSL für Verkehrsinformationssysteme sowie die Umsetzung und Darstellung des Prototyp-Framework für Verkehrsinformationssysteme. Implementierungsdetails werden im nächsten Kapitel besprochen. Es wird dabei versucht, die Konzepte möglichst generisch zu beschreiben, allerings lässt es sich nicht vermeiden, dass die Technologiewahl einige Aspekte beeiflusst. Auch was DSLs angeht, sollte man dieses Kapitel mehr als Erfahrungsbericht ansehen, da andere DSL unter Umständen völlig andere Anforderungen haben können. Daher sollte man vorher prüfen, welche Werkzeuge und Frameworks zum Einsatz kommen sollen (vgl. Grundlagen) und inwiefern sich die Konzepte übertragen lassen. Grundsätzlich sollte dieses Kapitel aber zumindest Ansatzpunkte liefern.

Die Konzeption erfolgte nicht vorab, sondern inkrementell und wurde in mehreren Iteration verfeinert. Es würden also im Laufe der Arbeit auch Aspekte geändert, hinzugefügt oder verworfen. Da sich im Laufe der Iterationen auch die Anforderungen geändert haben, sind hier auch Anforderungen gelistet, die in der Endfassung der eigentlichen Anforderungen weggefallen oder ersetzt wurden. Da diese Informationen aber die Arbeit beeinflusst haben, werden sie hier trotzdem aufgeführt.

## 4.1. Grundlegende Architektur

Die Anwendung besteht, wie Abbildung 4.1 zeigt, aus zwei Hauptteilen. Zum einen die Sprache an sich und zum anderen der Teil, der den Testprototyp generiert. Es war Anfangs noch nicht klar, wie generisch die Sprache sein sollte. Es gab zwei Möglichkeiten: Die erste Variante wäre die Sprache generisch zu gestalten, so dass man zwar entsprechende Komponenten für die Domain hat, aber prinzipiell alles sehr frei gestalten kann. Die zweite Variante wäre, dass man in der Sprache keinen direkten Einfluss mehr auf die Oberfläche hat, sondern anhand Eingabe, Ausgabe und gegebenenfalls anderer Parameter eine Oberfläche erstellt wird. Die ursprüngliche Idee war, es den Code für den Prototypen dynamisch zu generien. Dieses Vorgehen wäre das passende Konzept für Variante eins. Es stellte sich dann aber heraus, dass eigentlich Varinate zwei gewünscht wird. Entsprechend wurde der ursprüngliche Plan modifiziert. Anstatt die Anwendung dynamisch zu generieren, sollte diese weitgehend statisch sein, das heißt, dass kein eigener Code für jeden Prototyp erstellt wird, sondern nur eine Art Konfiguration.

Die Anwendung sollte damit also nun in zweimal zwei Teile gegliedert werden. Auf der einen Seite der Parser für die Syntax und der Generator, der die Koniguration für die Anwendung

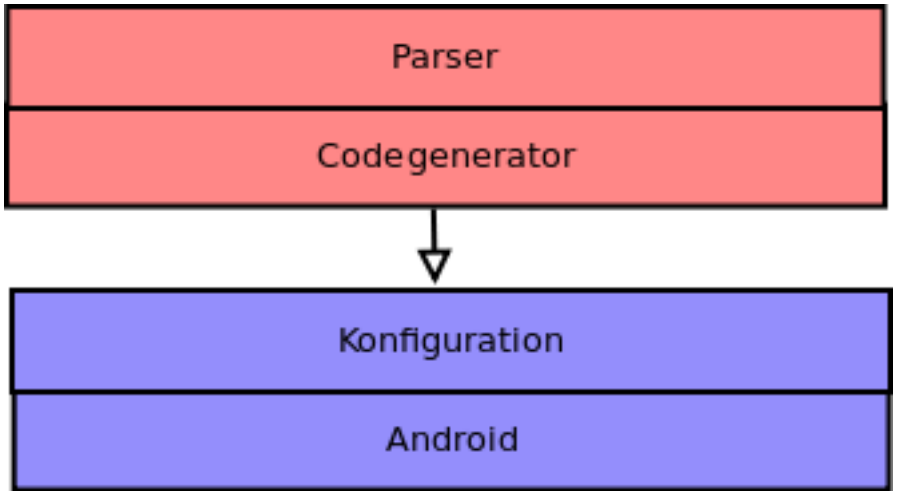


**Abbildung 4.1.:** Der rote Teil beschreibt den Parser und Generator, der blaue Teil bildet zusammen die Androidanwendung

erzeugt. Auf der anderen Seite die Konfiguration und das Framework, das zusammen den eigentliche Prototypen bilden. Es sollte also keine Oberfläche direkt erzeugt werden, da die Syntax der Sprache eine semantische Beschreibung der Oberfläche sein sollte. Es sollte also beschrieben werden, welche Elemente angezeigt werden. Der Pool an Elementen ist allerdings statisch und neue Elemente mit eigener Semantik können dabei nicht erzeugt werden. Diese müssen, falls benötigt, durch programmatische Erweiterung der Anwendung hinzugefügt werden. Der Vorteil ist, dass die Konfiguration nur eine dünne Schicht zwischen DSL und der eigentlichen Anwendung ist. Ein Austauschen der beiden Teile ist problemlos möglich. So könnte man beispielsweise die Xtext DSL durch eine Scala DSL ersetzen. Oder auch den Android Teil durch einen Windows Phone 7 Teil. Durch die Konfiguration verliert der Benutzer zwar Flexibilität, allerdings kann nur so die Sprache auf einem einfachen Level gehalten werden, die vollkommen unabhängig von der Plattform ist.

## 4.2. Die Oberfläche

Die Oberfläche besteht grundsätzlich aus zwei Teilen: Einem Eingabeteil und einem Ausgabeteil. Diese Teile bestehen gegebenenfalls aus mehreren Screens, die auch von der Konfiguration abhängig sein sollen.

### 4.2.1. Informationsmodell

In diesem Abschnitt werden die relevanten Typen und gegebenenfalls ihre Ein- und Ausgabe beschrieben.

#### Ort

Ein Ort ist für ein Verkehrsinformationssystem eine zentrale Eingabe. Es muss zumindest ein Startort bekannt sein. Meistens auch ein Zielort, da man in der Regel zu einem bestimmten Ort will. Der Ort kann dabei auf verschiedene Art und Weise ermittelt werden: Durch Eingabe des Namens oder Koordinaten des Orts oder durch Auswahl auf einer Landkarte. Für den Startort kann auch die Ermittlung via Standortsensoren wie GPS herangezogen werden. Für den Zielort können auch ein semantischer Begriff wie zum Beispiel Supermarkt, Kino oder Bank sinnvoll sein, wie es unter anderem die Online-Comminuty Qype [Qyp] anbietet. Diese bietet auch entsprechende Zusatzinformationen zu Geschäften. Solche Zusatzinformationen sind in der Ausgabe je nach Suchanfrage und Nutzertyp ebenfalls wünschenswert.

#### Zeit

Auch die Eingabe von Zeiten ist zentral für Verkehrsinformationssysteme. Typischerweise wissen wir, wann wir los wollen oder wann wir da sein müssen. Ebenso, wie viel Zeit wir an einem Zwischenhalt verbringen wollen. Die Zeiten sind auch in der Ausgabe wichtig; da die Eingabezeiten in der Regel nicht exakt eingehalten werden, müssen wir dabei von Hand auswählen, welche Zeitverschiebung für uns am ehesten akzeptabel ist. Optimalerweise werden in der Ausgabe auch direkt mögliche Verspätungen angezeigt.

#### Verkehrsmittel

Im Gegensatz zu Ort und Zeit ist die Auswahl des Verkehrsmittels meist optional. Meist ist es sinnvoll, hier alle Verkehrsmittel zu berücksichtigen, außer man will zum Beispiel Geld sparen, indem man Schnellzüge nicht berücksichtigt. Anders kann es mit Eigenschaften der Verkehrsmittel aussehen. Jemand, der im Rollstuhl sitzt, kann nur Verkehrsmittel berücksichtigen, die damit verträglich sind. Bei einem Zug sind heutzutage neben klassischen Ausstattungsmerkmale wie Speisewagen oder Schlafwagen auch Faktoren wie WLAN und Strom wichtig. Diese Ausstattungsmerkmale sollten neben der Anzahl der Sitzplätze in den

jeweiligen Klassen und weiteren Informationen zum Zug auch in der Ausgabe abrufbar sein. Eine weitere nützliche Funktion ist das Einstellen der Laufgeschwindigkeit, wie es zum Beispiel die VVS App unterstützt [VVS].

### 4.2.2. Eingabe

Ein wesentlicher Baustein der Arbeit ist die Eingabe von Werten. Prinzipiell soll es möglichst viele verschiedene Möglichkeiten geben Werte einzugeben. Dabei gibt es natürlich verschiedene Werttypen, bei denen nicht immer jedes Eingabeelement Sinn macht. Bei Eingabeelementen wie dem DatePicker ist klar, dass man mit ihnen nur Datumsangaben auswählen kann. Eine Checkboxliste kann man zwar prinzipiell schon zur Ortsauswahl nutzen, es macht in den meisten Fällen allerdings wohl keinen Sinn, da die Auswahl viel zu eingeschränkt ist, es sei denn, man hat die Auswahl schon zuvor entsprechend eingeschränkt. Ansonsten sind Eingabemöglichkeiten auch oft gut kombinierbar. So könnte man beispielsweise für die Ortsauswahl das Textfeld mit Autovervollständigung zusätzlich mit einem Button für die Auswahl von Orten auf der Landkarte versehen und nach der Auswahl den Ort in das Textfeld eintragen.

Der Prototyp soll zum einen mit einer Anzahl an eingebauten Standardeingabeelementen, die im Folgenden beschrieben werden, entwickelt werden. Zum anderen soll es auch möglich sein, eigene Eingabeelemente zu entwickeln um diese dann im Prototyp verwenden zu können.

In Abbildung 4.2 werden verschiedene Suchparameter auf einer Seite angezeigt. Weitere Varianten wären zum Beispiel eine Eingabe über einen Wizard.

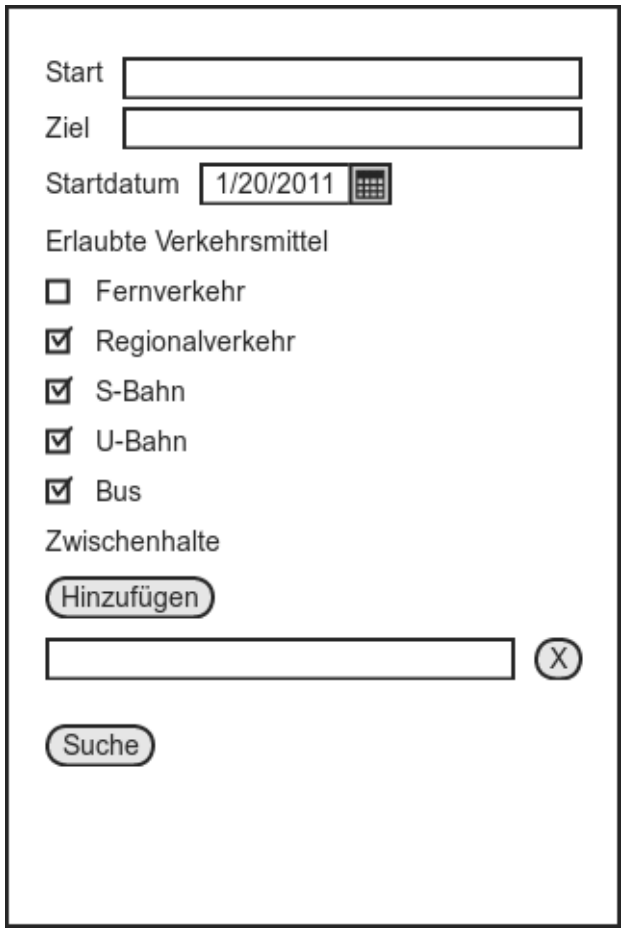


**Abbildung 4.2.:** Eine Variante des Eingabescreens.

#### Textfeld

Ein Textfeld lässt beliebige Texteingaben zu. Es kann damit sehr flexibel eingesetzt werden. Es eignet sich insbesondere für Werte, die nicht auf bestimmte Eingaben beschränkt sind, wie zum Beispiel für einen Ortsnamen. Weniger geeignet ist es für Eingaben mit bestimmtem Format wie zum Beispiel ein Datum oder Eingaben, bei denen die Eingabemenge beschränkt ist wie zum Beispiel die Liste von Zugtypen.

Im Prototyp ist ein Textfeld die einfachste Art der Eingabe.

#### Textfeld mit Autovervollständigung

Das Textfeld mit Autovervollständigung ist, wie der Name schon sagt, ein Textfeld, das automatische Textvervollständigung unterstützt. Dieser Mechanismus hat insbesondere durch Webbrowser und in jüngster Zeit auch Webanwendungen Popularität erreicht. In der einfachen Form werden nur Worte vervollständigt, die der Benutzer schon früher in dieses Textfeld eingegeben hat. Oft kommen die Vorschläge auch aus einer Datenbank, die alle möglichen Eingaben enthält. Ist diese Datenbank besonders groß, wie zum Beispiel eine Liste aller möglichen Bahnhöfe in Deutschland, beschränkt sich die Datenbasis auf besonders populäre Eingaben.

Im Prototyp soll die Autovervollständigung möglichst dynamisch gelöst werden. Dies soll mit einer Abfrage der Datenbasis nach jeder Wortveränderung realisiert werden.

#### Auswahl nach Landkarte

Die Auswahl auf einer Karte ermöglicht die grafische Auswahl von Orten. Für die Nutzer ist dies insbesondere dann sinnvoll, wenn er den genauen Ort nicht kennt, sondern nur dessen ungefähre geografische Lage. Bei einer großen Karte ist eine Zoom-Funktion zur schnellen Navigation wichtig. Auch eine Suche nach Orten ist eine sinnvolle Funktion. Sind Standortinformationen über Sensoren, verfügbar kann die Karte auch entsprechend zentriert werden und der aktuelle Standort besonders markiert werden.

Im Prototyp soll die Karte einfach aus einem grünen Hintergrund mit mehreren Orten bestehen. Da die Karte in ihren Maßen beschränkt sein soll, ist eine Zoom-Funktion nicht nötig. Auch die Erdkrümmung soll außer Acht gelassen werden. Die Auswahlfunktion soll wahlweise auch mit dem Textfeld beziehungsweise mit dem Textfeld mit Autovervollständigung kombiniert werden können. Die Karte wird dabei immer auf einem extra Screen angezeigt.

#### Auswahl aus Checkboxliste

Eine Checkbox ist ein Komponente der für die Auswahl boolescher Werte geeignet ist. Eine Liste davon wird im Prototyp dafür verwendet, um die bei der Suche zu beachtenden Verkehrsmittel auszuwählen. Da die Liste unter Umständen viel Platz auf dem Bildschirm einnehmen kann, bietet es sich an, sie ein- und ausklappbar zu machen oder die Eingabe auf einem extra Screen zu tätigen.

#### Liste aus Textfeldern

Die Liste von Textfeldern ist eine relativ spezielle Komponente. Im Ausgangszustand ist entweder ein Textfeld vorhanden, in diesem Fall kann dieses Textfeld auch nicht mehr entfernt werden, oder es muss erst dynamisch über einen Button hinzugefügt werden. Alle hinzugefügten Textfelder können auch wieder duch einen Button gelöscht werden.

Im Prototyp wird die Liste aus Textfeldern für Zwischenhaltestellen benutzt. Sie sollen aber auch für multiple Ortseingaben genutzt werden. In diesem Fall soll auch das Textfeld mit Autovervollständigung beziehungsweise mit der Kartenfunktion als Listeneintrag dienen können.

### 4.2.3. Ausgabe

Die Ausgabe hat im Vergleich zur Eingabe relativ wenige Konfigurationsoptionen. Eine Listendarstellung hat sich zur Ausgabe bewährt, da es selten der Fall ist, dass nur es ein eindeutiges Ergebnis gibt. Variable sind dabei die Daten, die in der Liste angezeigt werden. Typischerweise gibt es dann noch einen oder mehrere Screens mit Detailinformationen zu einem Listeneintrag.

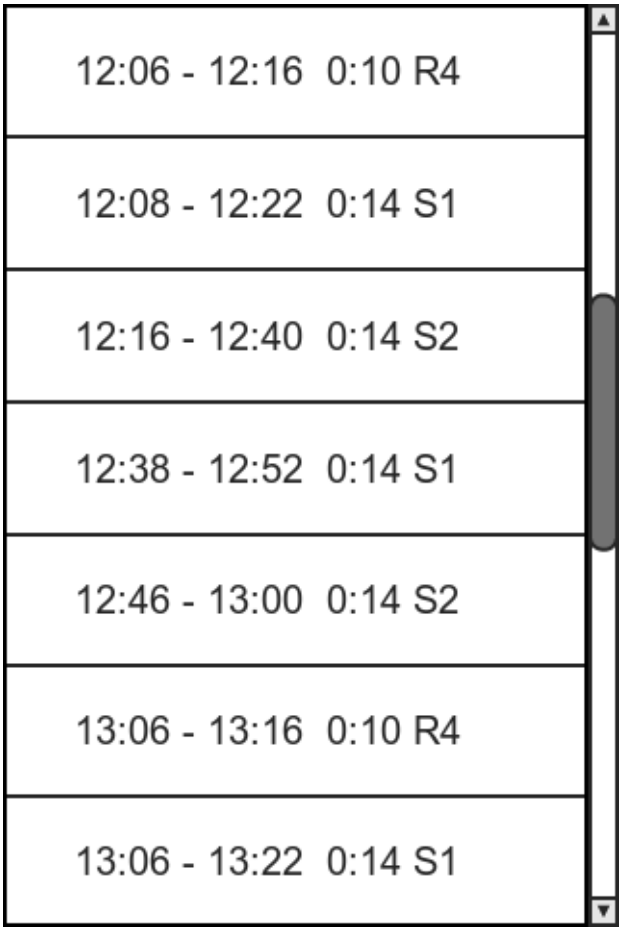


**Abbildung 4.3.:** Eine Variante des Ausgabescreens.

In Abbildung 4.3 wird eine einfache Ergebnisliste mit Zeiten und Linie angezeigt.

Die Abbildung 4.4 zeigt eine Detailseite einer gefundenen Verbindung. Hier werden alle möglichen Details angezeigt. In fortgeschritteneren Varianten könnten hier auch Operationen wie zum Beispiel Ticketkauf möglich sein.


**Abbildung 4.4.:** Eine Variante des Ausgabescreens.

### 4.2.4. Mock-Daten

Zur Demonstration der Anwendungen werden Mock-Daten benötigt. Dafür soll die Anwendung eine Schnittstelle zur Verfügung stellen. Diese soll programmatisch sein und Eingabedaten bekommen und daraus dann entsprechende Ausgabedaten zu generieren. Dieser Vorgang soll dabei austauschbar in der Architektur vorgesehen sein.

Eine Validierung der Eingabe kann im Zuge des Generierens der Ausgabe auch vorgenommen werden. So kann bei unvollständigen Eingaben ein Fehler geworfen werden, der den Nutzer zurück zur Eingabe bringt und dort die entsprechende Fehlermeldung anzeigt.

## 4.3. Benutzerparameter

Ein zentraler Punkt der Arbeit ist die Abbildung verschiedener Benutzertypen von Verkehrsinformationssystemen mit Hilfe des Prototypgenerators. Im Rahmen einer Fachstudie wurden schon einige Personas erarbeitet [AWZ10], optimalerweise sollte sich diese Liste aber erweitern lassen. Dies soll mittels sogenannter Benutzerparamter erfolgen. Diese drücken Eigenschaften aus, die der Benutzer des Prototyps haben soll und generiert entsprechend eine passende Oberfläche. Die Abstufung muss dabei zwar eigentlich nicht so hoch sein, aber um ausreichend Spielraum zu besitzen, wurde die Anzahl der Stufen auf 100 festgelegt. Für den zu erstellenden Prototyp sollen aber Abstufungen im ganzzahligen Prozentbereich auf jeden Fall ausreichen.

Für den Prototyp wurden dafür fünf Benutzerparameter festgelegt.

### 4.3.1. Führungsbedürfnis

Der Parameter Führungsbedürfnis steuert, wie sehr die GUI den Benutzer führen soll. Ein Benutzer mit hohem Führungsbedürfnis bevorzugt weniger Eingaben und diese auch eher nacheinander. Ein Nutzer der wenig Führungsbedürfnis hat, will dagegen eher möglichst viele Eingaben auf einem Screen haben und dabei die Freiheit haben, diese beliebig auszufüllen.

### 4.3.2. Leseaffinität

Der Paramter Leseaffinität gibt an, ob eher längere Erläuterungen oder kürzere Texte bevorzugt werden. Dies könnte zum Beispiel bestimmen, ob die Ausgabe eher kompakt oder ausführlich ist. Der Parameter beeinflusst zudem, ob Anzeigebereiche eher ein- und ausklappbar oder eher statisch sein sollen.

### 4.3.3. Texteingaben

Dieser Paramter bestimmt, ob der Benutzer Texteingaben mag, also eher textbasierte Eingabeelemente oder eher grafische bevorzugt werden. Dies kann zum Beispiel beeinflussen, ob der Benutzer das Datum über ein spezielles Widget eingibt oder dafür ein einfaches Textfeld benutzt werden soll. Das Textfeld mag für diesen Zweck zwar meist umständlicher sein. Es hat aber unter anderem den Vorteil, dass problemlos Copy und Paste benutzt werden kann.

### 4.3.4. Filter

Mit diesem Parameter bevorzugt der Benutzer programmseitiges Filtern. Anhand dieses Parameter wird bestimmen, ob eher viele Filtereingaben bevorzugt werden oder ob sich der Benutzer lieber auf ein paar grundlegende Filter beschränkt, um sich anschließend durch eine längere Ergebnisliste zu kämpfen und diese selber beim Lesen zu "filtern".

### 4.3.5. Zeit

Diese Parameter sagt aus ob, der Benutzer sich Zeit für Anfragen nimmt. Also ob eher kurze schnelle Abfragen gewünscht sind oder mehr Wert darauf gelegt wird, dass die Oberfläche möglichst umfangreiche Parameter für die Abfragen anbietet. Ein Benutzer, der wenig Zeit hat, will eher weniger Auswahlmöglichkeiten und diese auch möglichst kompakt dargestellt.

# 5. Realisierung des Prototyps

Dieses Kapitel beschreibt die praktische Umsetzung des Prototyps. Dabei werden die verschiedenen Komponenten beschrieben und konkrete Problemlösungen aufgezeigt. Es dient auch als Entwicklerdokumentation und setzt daher ein gewisses Grundverständnis der objektorientierten Programmierung [Boo04] sowie von der Sprache Java [Javb] und dem Umgang mit der Entwicklungsumgebung Eclipse voraus. Bei weiteren eingesetzten Frameworks und Bibliotheken werden entsprechende Verweise gesetzt, beziehungsweise kurz die Grundlagen erläutert und gegebenfalls durch Beispiele veranschaulicht. Quelltextausschnitte sind aus Platz- und Verständnisgründen teilweise gekürzt dargestellt.

In Abschnitt 5.1 werden kurz ein paar Fakten zur Entwicklung erläutert. Im Abschnitt 5.2 wird die Entwicklung der DSL anhand von Beispielen erklärt. Im Abschnitt 5.3 wird die Entwicklung des Android Frameworks für die Prototypen eingegangen. Dabei werden sowohl die Android Grundlagen erläutert, als auch auf die Schnittstellen der Arbeit eingegangen. Im letzten Abschnitt 5.4 gibt es einen kurzen Eindruck vom Ergebnis der Arbeit.

## 5.1. Allgemeines zur Entwicklung

Die Entwicklung erfolgt mit dem JDK 1.6.0_24 unter Ubuntu Linux 10.04. Als Entwicklungsumgebung wird Eclipse 3.6.2 mit dem Eclipse Plugin Xtext in der Version 1.0.2 und dem Eclipse Plugin Android Development Tools (ADT) in der Version 10.0.1 verwendet. Als Quellcode Konventionen werden die Standardeinstellungen von Eclipse benutzt, die weitgehend den Java Code Conventions [Java] entsprechen.

## 5.2. DSL

Am Anfang des DSL-Baus mit Xtext steht die Erstellung eines Xtext Projektes in Eclipse. Neben dem Projektnamen müssen der Name der Sprache, der typischerweise in der Konvention von Java-Paketen angegeben wird, und die Dateierweiterung der Quellcodedateien der neuen Sprache angegeben werden. Die Option „Create a generator project“ wird benötigt, da ein Generator-Projekt erzeugt werden soll, das für die Umwandlung des domänenspezifischen Quellcodes in eine Konfiguration für das Android Framework zuständig ist. Xtext erstellt nun drei Projekte:

- de.knittig.da.vis_da_prototype: Das Hauptprojekt. Es enthält die Grammatik für die DSL sowie zusätzliche Konfigurationen für Validierung, Scopes und Formatierung. Außerdem liegt hier das aus der Grammatik generierte EMF Modell, auf das geparste DSL abgebildet wird.
- de.knittig.da.vis_da_prototype.generator: Enthält das Template für die Generierung der Konfiguration.
- de.knittig.da.vis_da_prototype.ui: Enthält den Quellcode für die Generierung eines Eclipse Plugins für den DSL Editor.

### 5.2.1. Syntax

Die Syntax der Sprache wird durch die sogenannte Grammatiksprache von Xtext festgelegt. Dabei wird beschrieben, wie die konkrete Synatx aussieht und wie sie während des Parse-Vorgangs zu dem Modell abgebildet wird [Xtec].

Aus dieser Grammatiksprache werden drei Dinge generiert. Zum einen wird ein EMF Modell [EMF] generiert, dies ist ein standardisiertes Java Domainmodellklassen, bei dem Funktionalitäten wie Serialisierung, Abfrage und Validierung eingebaut sind. Zum anderen wird eine ANTLR Grammatik [ANTb] erzeugt, die die definierte Syntax parsen kann. Als letztes wird noch ein Eclipse Plugin erzeugt, das einen Editor für die Sprache bereitstellt.

```
grammar de.knittig.da.prototype.Vis with org.eclipse.xtext.common.Terminals

generate vis "http://www.knittig.de/da/prototype/Vis"

Model:
	travel+=Travel;

Travel:
	'travel' name=ID '{'
		features+=FeatureType (',' features+=FeatureType)*
	'}';

enum FeatureType:
	start | destination | startdate | stopdate | stopover
```

**Listing 5.1:** Einfache DSL-Definition mit Xtext

Das Listing 5.1 zeigt ein einfaches Beispiel der Grammatiksprache von Xtext. In der ersten Zeile wird die Grammatik eindeutig benannt und dabei von einer bestehenden Grammatik abgeleitet. Die `generate` Anweisung ist für die Generierung des EMF Metamodells nötig. Der erste Parameter gibt dabei den Paketname an, der zweite eine eindeutige Namesraum-URL. Nun beginnt die eigentliche Definition der Grammatik. Als erstes definieren wir ein Wurzelelement im Modell. Dies verweist dann auf den Typ `Travel`, der einen Block mit einem eindeutigen Namen beschreibt. `ID` ist dabei ein von der Grammatik `org.eclipse.xtext.common.Terminals` importiertes Element, das eine zusammenhängende Zeichenkette beschreibt. In dem Block wird eine Liste von dem Typ `FeatureType` erstellt.

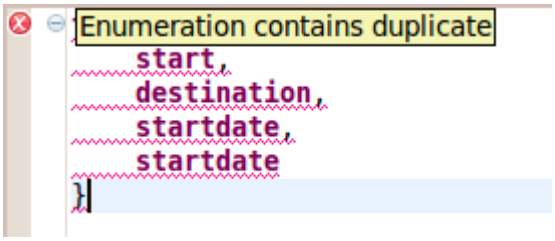


**Abbildung 5.1.:** Der durch einen Validator definierte Fehler in Eclipse mit dem DSL Eclipse Plugin

Dabei muss die Liste mindestens ein Element vom Typ `FeatureType` enthalten und danach null oder eine beliebige Anzahl, die jeweils durch ein Komma getrennt werden. `FeatureType` ist dabei eine einfache Aufzählung von Konstanten.

```
travel default {
    start, destination, startdate
}
```

**Listing 5.2:** Beispiel für die mit Xtext erstellte DSL

Mit Hilfe der Grammatik von Listing 5.1 kann dann die DSL 5.2 geparst werden. Mittels der generierten EMF Klassen kann man dann per Javacode prüfen, welche Elemente in der Aufzählung gesetzt sind. Die Grammatik ist aber noch verbesserungswürdig, da sie auch zulässt, dass ein Listenelement mehrfach in der Aufzählung auftaucht. Dies lässt sich nicht mit Hilfe der Grammatiksprache verhindern, stattdessen kann man in Xtext hierfür zusätzliche Validiererklassen, die in Java implementiert werden, in der Konfiguration angeben. Dort kann man eine Funktion wie in Listing 5.3 hinzufügen. Diese implementiert eine einfache Prüfung, ob Elemente in der Aufzählung doppelt vorhanden sind und wirft in diesem Fall einen Fehler, der dann, wie in Abbildung 5.1 zu sehen, im Editor des Eclipse Plugins entsprechend angezeigt wird. Wichtig ist dabei die `@Check` Annotation, um die Validierungsregel scharf zu schalten. Ansonsten wird die Methode zur Validierung von Xtext ignoriert. Zudem sollte das EMF-Modell bereits generiert sein, damit man die entsprechenden Domain-Klassen verwenden kann.

```
@Check
public void checkDupes(Travel travel) {
	Set<FeatureType> featuresInEnumeration = new HashSet<FeatureType>();
	for (FeatureType feature : travel.getFeatures()) {
		if (featuresInEnumeration.contains(feature)) {
			error("Enumeration contains duplicate", feature.ordinal());
		} else {
			featuresInEnumeration.add(feature);
		}
	}
}
```

**Listing 5.3:** Xtext Validator Beispiel

```
«IMPORT de::knittig::da::prototype::vis»

«DEFINE main FOR Travel-»
«FILE "de/knittig/da/MyConfiguration.java"»
package de.knittig.da;

import java.util.Arrays;

public class MyConfiguration implements Configurable {

    public Configuration getConfiguration() {
        Configuration configuration = new Configuration();
    «FOREACH features AS feature»
        configuration.set«feature.name.toLowerCase().toFirstUpper()»(true);
    «ENDFOREACH»
        return configuration;
    }
}
«ENDFILE»
«ENDDEFINE»
```

**Abbildung 5.2.:** Beispiel der Templateengine Xpand für die DSL aus Listing 5.2

Weitere Informationen dazu findet man in der Xtext Dokumentation [Xtec]. Hilfreich sind dazu außerdem Kenntnisse aus dem Bereich Compilerbau. Falls diese nicht vorhanden sind, kann man sich diese zum Beispiel über das bekannte „Drachenbuch“ [ASU86] aneignen.

### 5.2.2. Codegenerator

Nachdem man nun den Parser und Editor für die Sprache hat, ist der nächste Schritt einen Codegenerator zu entwickeln. Dazu bietet Xtext eine eigene Templateengine an: Xpand. Sie ist, im Gegensatz zu anderen Templateengines wie StringTemplate [Str], statisch typisiert, das heißt, dass Fehler beim Zugriff auf nicht vorhandene Klassen oder Felder bereits vor der Ausführung der Templateengine erkannt werden können. Xpand bezieht sich dabei direkt auf das von Xtext generierte EMF Modell und bietet auch Funktionen wie Autovervollständigung und Syntaxhervorhebung an.

Abbildung 5.2 zeigt ein einfaches Beispiel, welches die DSL von Listing 5.2 zu einer Java-Klasse umwandelt.

## 5.3. Android Framework

Für die Entwicklung von Androidanwendungen wird das Android SDK [Ande] verwendet, das neben den Bibliotheken auch einen Emulator zum Testen besitzt. Zudem gibt es damit auch die Möglichkeit eine Anwendung direkt auf dem Gerät zu testen gibt. Für Eclipse gibt

es das ADT (Android Development Tools) Plugin [Anda]. Alternativ gibt es zum Beispiel aber auch Android Unterütztung bei IntelliJ IDEA [Inta].

Für den Emulator wurde die Plattform Version 1.6 (API Level 4) zum Testen verwendet. Zwar ist diese Version für Smartphone-Betriebssysteme schon relativ alt, aber da für den Prototyp keine Funktionen der neueren Versionen benötigt werden, ist es aus Kompatiblitätsgründen sinnvoller eine ältere Version zum Testen zu verwenden, da diese zu neueren Version abwärtskompatibel sind. Als Entwicklungsgerät wurde das Google Nexus One [Nex] verwendet, auf dem die Version 2.2 läuft (API Level 8). Der Emulator hat eine Auflösung von 480x320 Bildschirmpunkten (HVGA), während das Nexus One eine Auflösung von 800x480 Bildschirmpunkten besitzt (WVGA).

Die Projektstruktur des Android Frameworks besteht aus drei Projekten:

- vis-da-model: Enthält Schnittstellen und Klassen der Ein- und Ausgabedaten. Siehe 5.3.2.
- vis-da-mockup: Enthält eine Mockup-Implementierungen der Datenquelle. Siehe 5.3.3.
- vis-da-prototype: Enthält den Code für die Interpretierung der Konfiguration und Oberflächenkomponenten. Siehte 5.3.4.

Vor der Erläuterung zu den einzelnen Projekten, werden in 5.3.1 kurz die nötigen Android Grundlagen zusammengefasst.

### 5.3.1. Android Grundlagen

Dieser Abschnitt erklärt kurz die wichtigsten Grundlagen, die für die Androidentwicklung notwendig sind. Weiterführende Informationen findet man unter anderem auf der offiziellen Seite für Android Entwickler [Andc].

#### Projekt Layout und Konfiguration

Wie in Abbildung 5.3 zu sehen, gibt es neben dem Quellcodeverzeichnis („src“) noch einige weitere Ordner in einem Android Projekt:

- res: In diesem Verzeichnis werden die Ressourcen gespeichert. Die Verzeichnishierarchie ist dabei fest vorgegeben. Siehe 5.3.1.
- gen: Dieses Verzeichnis enthält generierte Dateien. Standardmäßig ist dies nur die „R.java“, die Referenzen auf Ressourcen enthält.
- assets: Verzeichnis zur Ablage von beliebigen Daten, ohne Festlegung auf Verzeichnisse. Zugriff über den `AssetsManager`.

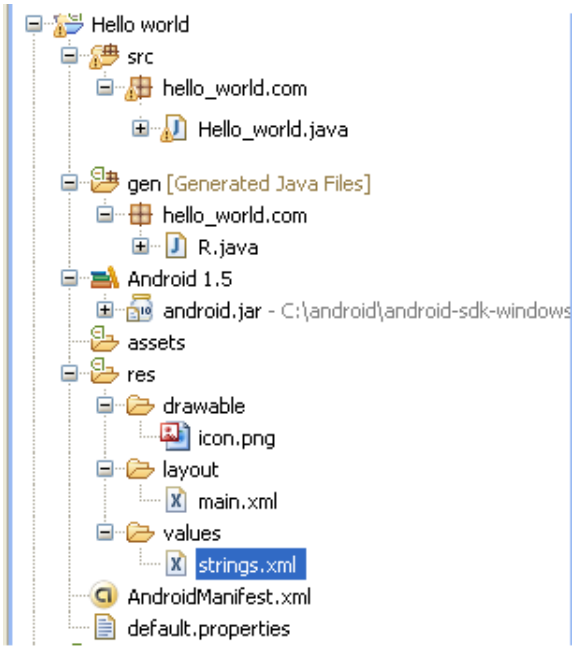


**Abbildung 5.3.:** Das Layout eines frisch erstellten Android Projektes in Eclipse

Daneben gibt es noch die Konfigurationsdatei „AndroidManifest.xml“. Hier werden neben Anwendungsname und -icon, auch alle Activities (siehe 5.3.1) sowie die Rechte, die die Anwendung benötigt, aufgezählt. Die Datei „default.properties“ ist dabei für das Buildwerkzeug Ant [Anta] vorgesehen.

### Activity

Eine Activity[1] ist ein Teil einer Androidanwendung. Weitere sogenannte Anwendungskomponenten sind Services, Content Providers und Broadcast Receivers, die aber für diese Arbeit nicht benötigt werden. Eine Activity repräsentiert einen einzelnen Bildschirm der Benutzeroberfläche einer Anwendung. Alle Activities müssen von der Klasse `Activity` abgeleitet werden. Eine Activity kann andere Activities aufrufen, dabei wird ein Intent übergeben (siehe 5.3.1). Von der neu gestarteten Activity kommt man dann, ohne weiteres zutun, mit dem Back-Button wieder zu der vorherigen Activity zurück. Auch wenn der Benutzer per Home-Button auf den Startbildschirm zurückspringt oder eine andere Anwendung in den Vordergrund springt, zum Beispiel bei einem Anruf, bleibem die letzten Activities der Anwendung aktiv.

[1] Deutsch: Aktivität

#### Intent

Ein Intent ist eine abstrakte Beschreibung einer Operation, die ausgeführt wird. Sie werden zur Übergabe von Daten zwischen Activities, Services und Broadcast Receivers verwendet. Ein Intent kann dabei nicht nur innerhalb einer Anwendung Daten austauschen, sondern auch anwendungsübergreifend. So kann zum Beispiel über einen Intent einfach innerhalb einer Anwendung beim Klick auf eine Adresse Google Maps aufgerufen werden, wo diese dann auf der Karte angezeigt wird.

#### Ressourcen

Mit Ressourcen bietet Android eine einfache Schnittstelle zum Zugriff auf Artefakte wie Bilder, String, Menüs und Farben. Sie werden innerhalb des „res“ Verzeichnisses (siehe Abbildung 5.3) gespeichert und beim Bauen der Anwendung über die „R.java“ Datei im „gen“ Verzeichnis zugänglich gemacht. Im Gegensatz zu Dateien im „assets“ Verzeichnis ist hier die Ordnerstruktur fest vorgegeben, das heißt Bitmaps müssen im Unterverzeichnis „drawable“ abgelegt werden.

### 5.3.2. Model

Das Projekt „vis-da-model“ kapselt Domainklassen und -schnittstellen für die Ein- und Ausgabe. Die zentrale Schnittstelle ist das `DataProvider` Interface. Es enthält nur die Methode aus Listing 5.4.

```
OutputData getResults(InputData inputData) throws IOException;
```

**Listing 5.4:** Die Transformationsmethode des DataProvider Interface

Diese Methode transformiert die Eingabe in Form von `InputData` in die Ausgabe `OutputData`. `InputData` und `OutputData` sind dabei beides Klassen, die eine `HashMap` kapseln, in der eine dynamische Menge an Werten gespeichert wird. Um einfacheren Zugriff zu gewähren, werden die Schlüssel für die Hashmaps als Stringkonstanten in den jeweiligen Klassen vorgehalten. Die zugehörigen Werte sind Arrays von `InputValues` für die `InputData` Klasse und Arrays von `DataItems` für die `OutputData` Klasse. Von diesen Basisklassen werden verschiedene Typen abgeleitet. Wichtig ist dabei, dass die Basisklassen, und damit auch alle abgeleiteten Klassen, serialisierbar sind, also das Interface `Serializable` implementieren. Dies ist nötig, um die Daten in der Android Anwendung zwischenspeichern zu können.

### 5.3.3. Mockup

Das Projekt „vis-da-mockup“ enthält die Imlementierung des `DataProvider` Interfaces sowie andere Implementierungen von Schnittstellen aus dem Modell Projekt. Es dient dazu die Eingabedaten in die Ausgabedaten zu transformieren. In diesem Fall wird die Datenbank SQLite [SQL] als Datenquelle verwendet, welche Android von Haus aus unterstützt.

### 5.3.4. Prototype Framework

Das Projekt „vis-da-prototype“ enthält das Android Framework für die Erstellung von Prototypen. Es gliedert sich dabei in mehrere Teile auf:

- Activities: Enthält verschiedene Bildschirme für die Anwendung. Siehe 5.3.4.
- Komponenten: Enthält verschiedene Komponenten für die Oberfläche der Anwendung. Siehe 5.3.4.
- Schnittstellen: Enthält übergreifende Schnittstellen und Domainklassen für die anderen Teile. Siehe 5.3.4.

#### Activities

Eine Activity beschreibt, wie in 5.3.1 erläutert, einen Bildschirm in einer Androidanwendung. Der Prototyp bestehe dabei aus relativ wenig Bildschirmen.

Der Wichtigste ist der Hauptbildschirm für die Abfrage von Verkehrsinformationen, auf dem alle Eingaben getätigt werden. Er setzt sich sehr variabel zusammen und wird von der Klasse `ExpandableListActivity` abgeleitet. Die Komponente `ExpandableList` ist eine Liste die ein- und ausklappbar ist. Die Klasse `ExpandableListActivity` ist folglich eine Activity mit einer `ExpandableList`, die den Bildschirm vollständig ausfüllt. Da unter Umständen nicht immer Komponenten dabei sind, die überhaupt ein- und ausklappbar sein sollen und manche sogar zwingend immer sichtbar sein sollen, können diese Komponenten an den Anfang oder ans Ende der Komponente gesetzt werden. Stattdessen neben der `ExpandableList` noch andere Komponenten auf dem Bildschirm zu platzieren funktioniert übrigens nicht: Die Komponenten scrollen dann nicht mit, was für den Benutzer ein unschönes Erlebnis ist.

Für die Ausgabe gibt es zwei relevante Bildschirme: Einen zur Anzeige von Ergebnissen und einer für die Detailansicht. Letzterer kann auch optional sein, dies macht allerdings in der Praxis wenig Sinn, da meist nicht alle Informationen in der Ergebnisliste untergebracht werden können. Im Augenblick sind beide Bildschirme als `ListActivity` implementiert, es könnte allerdings auch Varianten geben, in denen der Ergebnisbildschirm eine `ExpandableListActivity` ist und der Detailbildschirm eine normale `Activity` mit individuellen Komponenten.

Zwischen den verschiedenen Bildschirmen wird über Intents kommuniziert. Zwischen Ein- und Ausgabe wird zudem die Implementierung des `DataProvider` aus dem Mockup Projekt aufgerufen und die Eingaben zu in die Ausgaben transformiert.

#### Komponenten

Komponenten bezeichnen in diesem Fall grafische Komponenten. In Android sind alle grafischen Komponenten von der Klasse `View` abgeleitet. Die wichtigsten wurden in Abschnitt 4.2.2 beschrieben, davon sind einige in Android Standardkomponenten, wie die `TextView`, die ein Textfeld repräsentiert, oder die `AutoCompleteTextView`, die ein Textfeld mit Autovervollständigung abbildet. Für die Karten-Komponente gäbe es zwar eine Komponente, die aber die reale Welt abbildet und somit nicht für diesen Zweck einsetzbar ist. Stattdessen wurde eine eigene Karten-Komponente erstellt, die zugleich auch die aufwendigste Komponente ist. Der erste Versuch die Karte als Bitmap zu generieren und diese scoll- und anklickbar zu machen schlug fehl, da dafür nicht genug Speicher vorhanden war. Im zweiten Versuch wurden einfach Punkte auf eine View mit grünem Hintergrund eingezeichnet und diese beim Scrollen korrekt verschoben. Dazu muss die Methode `onTouchEvent` mit dem Parameter `MotionEvent` implementiert werden. Der Parameter enthält dabei, ähnlich wie bei einer Maus, die Angabe, wann der Finger an das Touchdisplay angelegt wurde, wann er gezogen würde und wann er wieder vom Display entfernt würde. Damit lässt sich dann die Verschiebung des Kartenausschnittes berechnen. Weitere Komponenten sind der `DateTimePicker` für das Auswählen von Datums- und Zeitwerten und die `EditTextList`, die eine erweiterbare Liste von Textfelden darstellt.

#### Schnittstellen

In diesem Teil des Frameworks werden übergreifende Interfaces und Domainklassen abgelegt, hauptsächlich zur Beeinflussung und Erweiterung der Generierung der Oberfläche, also der Verschmelzung der Anwendungsteile von 5.3.4 und 5.3.4. Wichtig ist hier die abstrakte Klasse `AbstractControlCreator`. In ihr gibt es verschiedene Methoden für die abstrakten Komponenten (siehe 4.2.1). Die Auswahl soll unter Beachtung der Benutzerparameter (siehe 4.3) erfolgen. Zu beachten ist allerdings, dass diese Arbeit auf Grund der Komplexität der Aufgabe keinen ausgefeilten Algorithmus dafür enthält, sondern diese Möglichkeit für nachfolgende Arbeiten offen hält.

## 5.4. Entwicklungsergebnis

Das Ergebnis der Arbeit ist ein Eclipse Plugin für die Entwicklung der DSL für Verkehrsinformationssysteme für mobile Anwendungen. Mit ihm können verschiedene Testszenarien abgebildet werden und es steht Erweiterungen offen. Wie das ganze dann aussehen kan sieht man in Abbildung 5.4.

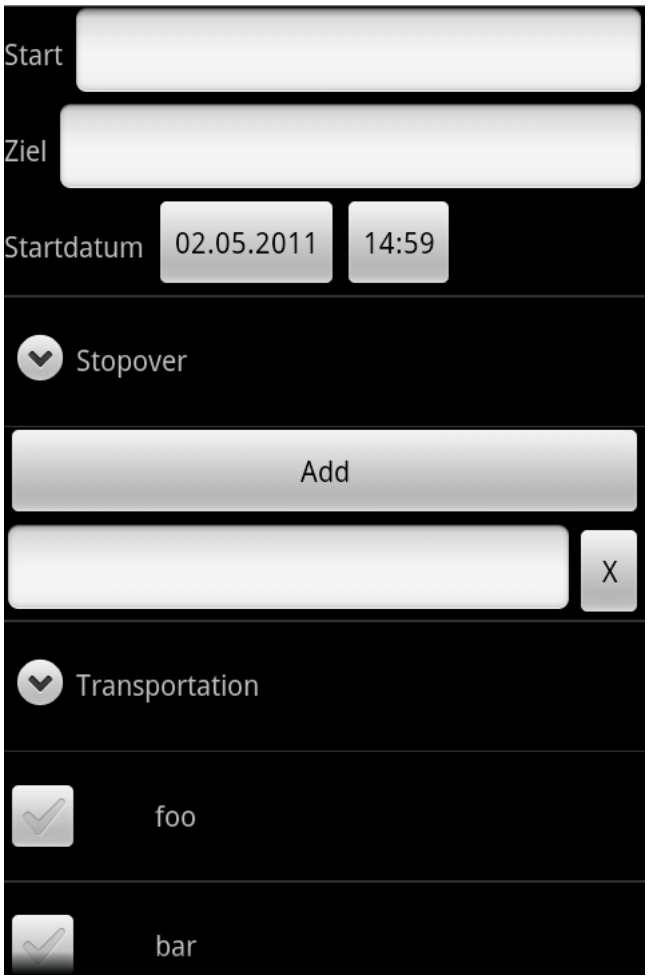


**Abbildung 5.4.:** Bildschirm des Prototyp in der Androidanwendung

# 6. Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde ein Codegenerator für die Erstellung von Prototypen für mobile Anwendungen im Rahmen des IP-KOM-ÖV Projektes entwickelt. Mit Hilfe der Prototypen sollen verschiedene Testszenarien von Verkehrsinformationssysteme auf mobile Geräten durchgespielt werden.

Dafür wurden, nach einer längeren Phase der Anforderungssammlung, verschiedene mobile Plattformen sowie Sprachen und Werkzeuge für die Erstellung von DSLs untersucht. Die Auswahl der mobilen Plattform gestaltete sich als einfach. Die Android Plattform ist dank ihrer Offenheit und bereits größeren Verbreitung zur Zeit die beste Wahl. Die Wahl der passenden Sprache beziehungsweise des passenden Werkzeuges für die Erstellung der DSL gestaltete sich in diesem Fall schwieriger. Anhand von einer kleinen Beispiel DSL wurden die verschiedenen Sprachen und Werkzeuge getestet. Hierbei machten alle Kandidaten eine gute Figur. Mit Xtext wurde im Nachhinein betrachtet hier ebenfalls eine brauchbare Wahl getroffen.

In der Konzeption wurde der Entwurf für die Prototypen erstellt. Dabei wurden die nötigen Konzepte von Verkehrsinformationssysteme und die passenden grafischen Komponenten für die Oberfläche beschrieben. Ebenfalls beschrieben wurden die Schnittstellen für Mockup-Datenquellen und Benutzerparameter. Benutzerparameter sollen in Zukunft dazu dienen, die Oberfläche an verschiedene Nutzertypen anzupassen.

Der größte Teil der Arbeit umfasst die prototypische Implementierung des Codegenerators und des dazugehörigen Frameworks für Android. Dazu wurde eine DSL-Syntax mit Hilfe der Xtext Grammatiksprache festgelegt, mit der, neben einem Parser, auch ein Editor für die DSL erstellt wird. Aus der DSL wird dann eine Konfiguration generiert, die zusammen mit dem Android Framework eine Androidanwendung ergibt. Diese Androidanwendung bildet dann ein Testszenario des Projektes IP-KOM-ÖV für Verkehrsinformationssysteme auf mobilen Geräten.

# 7. Ausblick

Mit dem Themen modellgetriebene Softwareentwicklung und mobile Plattformen behandelt diese Arbeit, gleich zwei Themen in denen (hoffentlich) noch die nächsten Jahre noch einiges an Innovation kommen wird.

Das Thema modellgetriebene Softwareentwicklung wird auch in den nächsten Jahren noch ein großes Thema sein. Die Entwicklung von Werkzeuge wie Xtext und MPS hat in den letzten Jahren deutlich an Fahrt aufgenommen. Zudem werden jedes Jahr neue Sprachen entwickelt, die innovative Ansätze mitbringen und oft den Bau von DSLs gut unterstützen. Zwar setzen sich die meisten davon nicht durch, aber im Zuge dessen ziehen auch die großen Sprachen wie Java, C# und C++ mit neuen Sprachfunktionen nach. Allerdings sind die Hürden für Einsteiger immer noch deutlich vorhanden, so dass davon auszugehen ist, das sich das Tempo bei der Verbreitung der modellgetriebene Softwareentwicklung nicht wesentlich beschleunigt.

Bei mobilen Plattformen gibt es gleich eine ganze Reihe von Trends. Hier die wichtigstens Trends, die insbesondere auch für Verkehrsinformationssysteme interessant sind.

**Tablets** Mit der Einführung des iPads Anfang 2010 ist Apple ein weiterer großer Wurf gelungen. Nachdem Tablets jahrelang ein Nischenmarkt waren, übertrifft das iPad im ersten Jahr die meisten Erwartungen. Noch hat es keine andere Plattform geschafft diese Dominanz zu brechen, aber es dürfte wohl nur eine Frage der Zeit sein bis sich ernsthafte Konkurrenz für das iPad herausbildet. Nachdem Android diese Geräteklasse mittlerweile auch offiziell unterstützt sind sie heißer Kandidat das Rennen zu machen. Wie Verkehrsinformationssysteme optimal auf Tablets angepasst werden können, dürfte also auf jeden Fall eine Zukunftsfrage sein.

**Mobiles Bezahlen** Statt am Schalter in der Schlange seine Fahrkarte zu lösen, werden wir vielleicht schon in naher Zukunft überall bequem mit dem Handy das Ticket bezahlen. Vereinzelt werden zwar schon Handytickets angeboten, doch das soll durch Near Field Communication, abgekürzt NFC, noch einfacher werden. Denn diese Technik ermöglicht unter anderem sichere und berührungslose Bezahlung. Android unterstützt NFC in der neusten Version bereits. Das Gerücht geht um das auch das nächste iPhone NFC unterstützen wird und Nokia hat angekündigt alle künftigen Smartphones mit NFC-Unterstützung auszuliefern.

**Augmented Reality** Mit Augmented Reality[1], abgekürzt AR, werden Bilder durch spezielle Programme durch Einblendungen um Zusatzinformationen angereichert. Zum Beispiel Informationen zu Sehenswürdigkeiten oder der Weg zu einem eingegebenen Ziel. Mittlerweile ist die Haardware von Smatphones stark genug und gibt es einige auf AR-Technologien spezialisierte Hersteller, die Versuchen zusammen mit Partnern Inhalte für AR aufzubereiten.

**3D** LG hat bereits den die ersten Android-Geräte mit 3D Display demonstriert und ein SDK für 3D ANwendungen zur Verfügung gestellt. Mit der 3D Kamera kann man zudem auch 3D Bilder und Videos aufnehmen. Zwar sind 3D Anwendungen bisher hauptsächlich im Unterhaltungsbereich, aber auch hier ist potenziall da das durch diese Technik ernsthafte Anwendungen bereichert werden können.

[1]Deutsch: erweiterte Realität

# A. Anhang

## A.1. Testfälle

Diese Testfälle sollen vom Prototype Generator abgedeckt werden. Der „*“ bedeutet dabei das mehrere oder auch kein Typ dieser Ein- oder Ausgabe möglich sind. Ein „?“ steht für kein oder ein Ein- oder Ausgabeelement. Das „+“ Für ein oder mehrere Ein- oder Ausgaben.

### A.1.1. Verbindung

Eingabe

- Startort
- Zielort
- Zwischenhalt*
- entweder Abfahrts- oder Ankunftszeit
- erlaubte Verkehrsmittel*

Ausgabe

- Verbindung*
  - Teilstrecke*
    - Abfahrtszeit
    - Ankunftszeit
    - Verkehrsmittel

### A.1.2. Umstieg

Eingabe

- aktuelles Vehikel
- Zielvehikel+
- Attribut des Umsteigeorts*

Ausgabe

- Umsteigehaltestelle*

### A.1.3. Reiseangebote

Eingabe

- Startort
- Zielort
- Maximalpreis

Ausgabe

- Verbindung*

### A.1.4. Einfache Verbindung

Eingabe

- Startort
- Zielort
- Startzeit

Ausgabe

- Verbindung*

### A.1.5. Platzreservierung

Eingabe

- Platztyp
- Klasse
- Nähe zum Speisewagen?

Ausgabe

- möglicher Sitz-/Liegeplatz*

Zusätzlich benötigte Eingaben

- Um die Eingaben oben zu spezifizieren, wird auch eine Verbindung bzw. ein konkretes Vehikel benötigt

### A.1.6. Billigstes Angebot mit anschließender Taxi-Buchung

Eingabe

- Taxi mit Anreise (des Taxis)?
- Taxi mit Rollstuhlbeförderung?

Ausgabe

- Angebot*

Zusätzlich benötigte Eingaben

- eine Verbindung wird benötigt, an die die Taxifahrt angeschlossen werden soll

biblatex

# Literaturverzeichnis


[Anda] ADT Plugin for Eclipse. URL http://developer.android.com/sdk/eclipse-adt.html. (Zitiert auf Seite 47)

[Andb] Android. URL http://www.android.com/. (Zitiert auf Seite 15)

[Andc] Android Developers. URL http://developer.android.com/. (Zitiert auf Seite 47)

[Andd] Android Market. URL http://market.android.com/. (Zitiert auf Seite 13)

[Ande] Android SDK. URL http://developer.android.com/sdk/index.html. (Zitiert auf Seite 46)

[Anta] The Apache Ant Project. URL http://ant.apache.org/. (Zitiert auf Seite 48)

[ANTb] Parsergenerator ANTLR. URL http://www.antlr.org/. (Zitiert auf den Seiten 22 und 44)

[app] applause. URL http://code.google.com/p/applause/. (Zitiert auf Seite 19)

[ASU86] A. V. Aho, R. Sethi, J. D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986. (Zitiert auf Seite 46)

[AWZ10] O. Alkis, C. Wimmer, F. Zoabi. *IT-Einsatzszenarien zur interaktiven Fahrgastunterstützung*. Fachstudie, Universität Stuttgart, 2010. (Zitiert auf den Seiten 9 und 40)

[Bad] Bada. URL http://www.bada.com/. (Zitiert auf Seite 16)

[Blaa] BlackBerry Developer Zone. URL http://us.blackberry.com/developers/. (Zitiert auf Seite 15)

[Blab] BlackBerry OS. URL http://www.blackberryos.com/. (Zitiert auf Seite 15)

[Booo4] G. Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004. (Zitiert auf Seite 43)

[Che] Günstige Android-Geräte. URL http://geizhals.at/deutschland/?cat=umtsover&xf=148_Android&sort=p. (Zitiert auf Seite 17)

[Coc] Cocoa. URL http://developer.apple.com/technologies/mac/cocoa.html. (Zitiert auf Seite 15)

[Dal] Code and documentation from Android's VM team. URL http://code.google.com/p/dalvik/. (Zitiert auf Seite 15)

[DBN] DB Navigator. URL http://www.bahn.de/p/view/buchung/mobil/mobile-apps.shtml. (Zitiert auf den Seiten 13 und 25)

[DKW10] A. Dridiger, M. Knittig, S. Wokusch. *Modellgetriebene und generative Benutzungsschnittstellen*. Fachstudie, Universität Stuttgart, 2010. (Zitiert auf den Seiten 7 und 19)

[Ecl] Eclipse. URL http://www.eclipse.org/. (Zitiert auf Seite 22)

[EMF] Eclipse Modeling Framework Project (EMF). URL http://www.eclipse.org/modeling/emf/. (Zitiert auf den Seiten 22 und 44)

[Fea] Definition Feature Phone. URL http://www.phonescoop.com/glossary/term.php?gid=310. (Zitiert auf Seite 13)

[Fow04] M. Fowler. Closure, 2004. URL http://martinfowler.com/bliki/Closure.html. (Zitiert auf Seite 20)

[Fow05] M. Fowler. Language Workbenches: The Killer-App for Domain Specific Languages?, 2005. URL http://martinfowler.com/articles/languageWorkbench.html. (Zitiert auf den Seiten 19 und 21)

[Hob00] E. Hobst. *Qualität von Verkehrsinformationssystemen*. Diplomarbeit, Universität Stuttgart, 2000. (Zitiert auf Seite 24)

[HPP] HP attackiert Apple mit Tablet-PC und Smartphones. URL http://www.welt.de/wirtschaft/webwelt/article12498962/HP-attackiert-Apple-mit-Tablet-PC-und-Smartphones.html. (Zitiert auf Seite 16)

[Inta] IntelliJ IDEA - Free IDE for Google Android. URL http://www.jetbrains.com/idea/features/google_android.html. (Zitiert auf Seite 47)

[Intb] Intentional Software. URL http://intentsoft.com/. (Zitiert auf Seite 19)

[iOSa] iOS Developer Program. URL http://developer.apple.com/programs/ios/. (Zitiert auf Seite 15)

[iOSb] iOS Human Interface Guidelines. URL http://developer.apple.com/library/ios/#DOCUMENTATION/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html. (Zitiert auf Seite 15)

[iph] iphonical. URL http://code.google.com/p/iphonical/. (Zitiert auf Seite 19)

[IPK] IP-KOM-ÖV. URL http://www.ip-kom.net/. (Zitiert auf Seite 7)

[IPK10] Standardisierungs-Forschungsprojekt 19P10003 IP-KOM-ÖV Gesamtvorhabensbeschreibung, 2010. (Zitiert auf den Seiten 7 und 9)

[Java] Code Conventions for the Java Programming Language. URL http://java.sun.com/docs/codeconv/html/CodeConventions.doc.html. (Zitiert auf Seite 43)

[Javb] Java. URL http://www.java.com/. (Zitiert auf Seite 43)

[JLo6] H. L. Jochen Ludewig. *Software Engineering - Grundlagen, Menschen, Prozesse, Techniken*. dpunkt.verlag, 2006. (Zitiert auf Seite 17)

[LY99] T. Lindholm, F. Yellin. *Java(TM) Virtual Machine Specification, The (2nd Edition)*. Prentice Hall PTR, 2 edition, 1999. (Zitiert auf Seite 21)

[Mae] Maemo. URL http://maemo.org/. (Zitiert auf Seite 16)

[Mee] Meego. URL http://meego.com/. (Zitiert auf Seite 16)

[Met] MetaEdit+. URL http://www.metacase.com/mep/. (Zitiert auf Seite 19)

[Mob] Moblin. URL http://moblin.org/. (Zitiert auf Seite 16)

[MPS] Jetbrains MPS. URL http://www.jetbrains.com/mps/. (Zitiert auf Seite 23)

[Nex] Nexus One Details. URL http://www.google.com/phone/detail/nexus-one. (Zitiert auf Seite 47)

[Nor10] P. Northam. Introduction to Bada, 2010. URL http://www.amiando.com/eventResources/h/B/4LHaCKJ6ye6mu3/Phil_Northam_Introduction_to_bada.pdf. (Zitiert auf Seite 16)

[Obj] Objective-C. URL http://developer.apple.com/documentation/Cocoa/Conceptual/ObjectiveC/. (Zitiert auf Seite 15)

[Oef] Oeffi. URL http://oeffi.schildbach.de/. (Zitiert auf Seite 27)

[OHA] Open Handset Alliance. URL http://www.openhandsetalliance.com/. (Zitiert auf Seite 15)

[Per10] P. Perrotta. *Metaprogramming Ruby*. Pragmatic Bookshelf, 1st edition, 2010. (Zitiert auf Seite 20)

[Pla] BlackBerry PlayBook. URL http://us.blackberry.com/playbook-tablet/. (Zitiert auf Seite 15)

[Pus] push&ride. URL http://www.pushandride.de/. (Zitiert auf Seite 28)

[Qyp] Qype. URL http://www.qype.com/. (Zitiert auf Seite 33)

[Rea] Realaxy ActionScript Editor. URL http://www.realaxy.com/. (Zitiert auf Seite 23)

[Ruba] Programmiersprache Ruby. URL http://www.ruby-lang.org/. (Zitiert auf Seite 20)

[Rubb] Webframework Ruby On Rails. URL http://rubyonrails.org/. (Zitiert auf Seite 20)

[Rubo8] Webframework Ruby On Rails, 2008. URL http://www.artima.com/weblogs/viewpost.jsp?thread=223054. (Zitiert auf Seite 20)

[Sma] Definition Smartphone. URL http://www.phonescoop.com/glossary/term.php?gid=131. (Zitiert auf Seite 13)

[Sma11] comScore Reports February 2011 U.S. Mobile Subscriber Market Share, 2011. URL http://www.comscore.com/Press_Events/Press_Releases/2010/4/comScore_Reports_February_2010_U.S._Mobile_Subscriber_Market_Share. (Zitiert auf den Seiten 13, 14, 16 und 17)

[SQL] SQLite. URL http://www.sqlite.org/. (Zitiert auf Seite 49)

[Str] StringTemplate. URL http://www.stringtemplate.org/. (Zitiert auf den Seiten 18 und 46)

[Sym] Symbian. URL http://symbian.nokia.com/. (Zitiert auf Seite 16)

[TIO] TIOBE Programming Community Index. URL http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. (Zitiert auf Seite 17)

[TSo7] S. E. u. A. H. Thomas Stahl, Markus Völter. *Modellgetriebene Softwareentwicklung*. Dpunkt.verlag Gmbh, 2007. (Zitiert auf Seite 18)

[VBN] Windows Phone 7-Apps mit VB.NET. URL http://www.aboutdotnet.de/post/Windows-Phone-7-Apps-mit-VB-NET.aspx. (Zitiert auf Seite 16)

[VVS] VVS App. URL http://www.vvs.de/fahrplan/mobilefahrplanauskunft/vvs-app/. (Zitiert auf den Seiten 26 und 34)

[web] webOS Developer Center. URL https://developer.palm.com/. (Zitiert auf Seite 16)

[Win10] Microsoft: "No Windows Phone 7 upgrade for Windows Mobile 6.x devices", 2010. URL http://apcmag.com/microsoft-no-windows-phone-7-upgrade-for-windows-mobile-6x-devices.htm. (Zitiert auf Seite 13)

[Xtea] ARText - Driving developments with TMF Xtext. URL http://www.eclipsecon.org/summiteurope2009/sessions?id=988. (Zitiert auf Seite 22)

[Xteb] Xtext. URL http://www.eclipse.org/xtext/. (Zitiert auf Seite 22)

[Xtec] Xtext Documentation. URL http://www.eclipse.org/Xtext/documentation/. (Zitiert auf den Seiten 44 und 46)

[Xted] Xtext Seite von Itemis. URL http://xtext.itemis.com/. (Zitiert auf Seite 22)

Alle URLs wurden zuletzt am 02.05.2011 geprüft.

# Abbildungsverzeichnis



# Verzeichnis der Listings

**Erklärung**

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

---

(Markus Knittig)