HOCHSCHULE DER MEDIEN

Stadtplandienst für mobile Endgeräte

Praktikum Softwaretechnik 2

Betreuer: Prof. Dr. Johannes Maucher

Wintersemester

2005	2006
Barbara Vogelmann	Michael Gerlinger
13381	13362

Medieninformatik

Inhalt

1 Einleitung

1.1 Motivation

Mittlerweile gibt es immer mehr Dienste für mobile Applikationen. Da uns dieses Thema interessierte, recherchierten wir ein wenig und fanden eine Lücke.
Es gab Dienste, die per SMS eine Wegbeschreibung verschickt haben und ähnliches, aber wir fanden keinen Stadtplandienst für eine mobile Applikation. Deswegen haben wir uns Gedanken darüber gemacht, warum eigentlich nicht?
Gibt es dafür technische oder betriebswirtschaftliche Gründe?
Oder liegt es an der (noch) zu langsamen Datenübertragung? Vielleicht dauert der Aufbau der Seiten zu lang oder ist die Displaygröße, Auflösung oder die Farben nicht gut genug, um Details der Karte erkennen zu können?
Darüber hinaus wollten wir uns auch noch mit dem Thema XHTML MP beschaffen, und herausfinden, was mit diesem XHTML-Standard möglich ist.
Zum Schluss wollen wir dann noch einen Usability Vergleich zwischen dem Stadtplandienst für das Handy sowie für das Web durchführen, um obige Fragen evtl. besser beantworten zu können.
Dazu wollen wir dann sowohl die mobile Applikation als auch die Webapplikation bauen (mehrkanalfähig), damit der Vergleich auch wirklich ein Vergleich ist, das bedeutet dass die Umgebungsparameter möglichst gleich sind.

1.2 Selbstdefinierte Anforderungen

Wir hatten eine ganze Reihe an Anforderungen an uns selbst, was wir alles umsetzen wollten.

- Erstellen eines Stadtplandienstes auf Basis von xHTML MP und JSP
- Erstellen eines Webfrontends auf Basis von HTML und JSP
- Mehrkanalfähig über Browserweiche
- Usability Vergleich

2 Technische Daten für das Nokia 6600

Für unsere mobile Applikation haben wir uns das Nokia 6600 herausgesucht, um darauf unsere Applikation zu testen. Dafür gab es mehrere Gründe:

- es ist ein Handy aus einer „älteren" Generation. Wenn die Applikation hier läuft, läuft sie wahrscheinlich auch gut auf neueren Modellen (was dann auch so war)
- die Farbauflösung ist noch auf dem unteren Level angesetzt mit 65 536 Farben.
- Es stand uns zur Verfügung
- Das Betriebssystem Symbian OS

2.1 Auszüge aus der Spezifikation für das Nokia 6600

2.1.1 Display und Benutzeroberfläche

Besonders großes, helles Aktivmatrix-Farbdisplay
65.536 Farben
176 x 208 Pixel
Grafische Benutzeroberfläche, kann vom Nutzer individuell gestaltet werden (Auswahl an Themen)
Joystick mit 5-Wege-Navigation
Betriebssystem: Symbian OS 7.0s

2.1.2 Browser

Erweiterter XHTML-Browser
Elektronische Brieftasche: vertrauliche Daten speichern (z. B. Passwörter und Kreditkarten-Daten) und komfortabel automatisierte Online-Transaktionen

2.1.3 Datenübertragung

Bis zu 40,2 KBit/s in GPRS-Netzen
Bis zu 43,2 KBit/s in HSCSD-Netzen

3 Funktionsumfang

Ortseingabe
Eingabe der Adresse über Ort oder PLZ

Doppelte / nicht eindeutige Adressen
Auswahlliste erscheint bei nicht eindeutiger Ortsangabe

Fehlerseite
Fehlerhandling auf einer Error-Seite

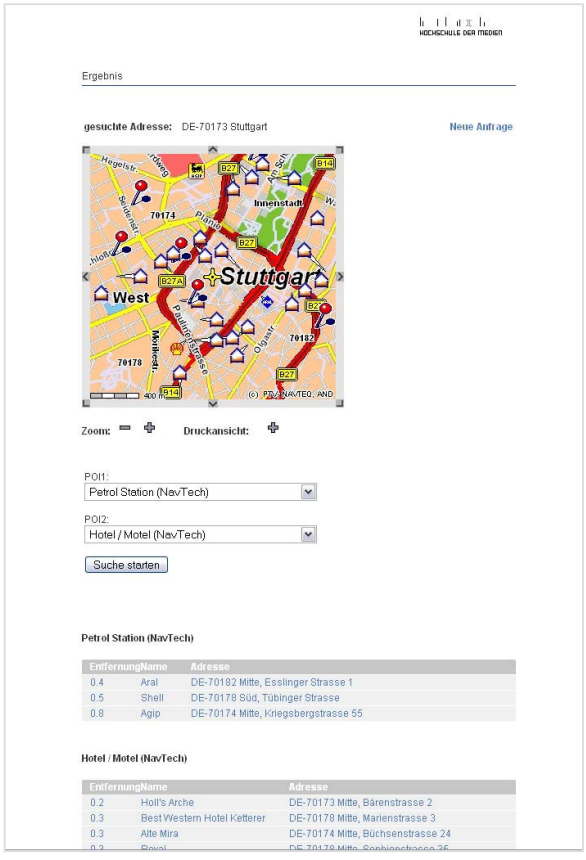
ZoomIn, ZoomOut
Hinein- und Herauszoomen in 25er Schritten
Bewegen der Karte in alle Richtungen
Nur Webfrontend

Druckansicht
größere Kartenansicht mit POI-Liste

POIs
1 (2 im Webfrontend) Kategorien auswählbar
In der mobilen Applikation über extra Zwischenseite neue POI-Kategorie wählbar
Im Webfrontend auf der Ergebnisseite 2 neue POIs auswählbar

4 Anzeigeseiten

4.1 Webfrontend



4.1.1 Funktionalitäten der Web Applikation

- 2 POI-Kategorien auswählbar
- POI Kategorien direkt auf Ergebnisseite neu auswählbar
- Druckansicht
- Bewegen der Karte in alle Richtungen möglich
- Kartenausschnitt größer

4.2 Mobile Applikation

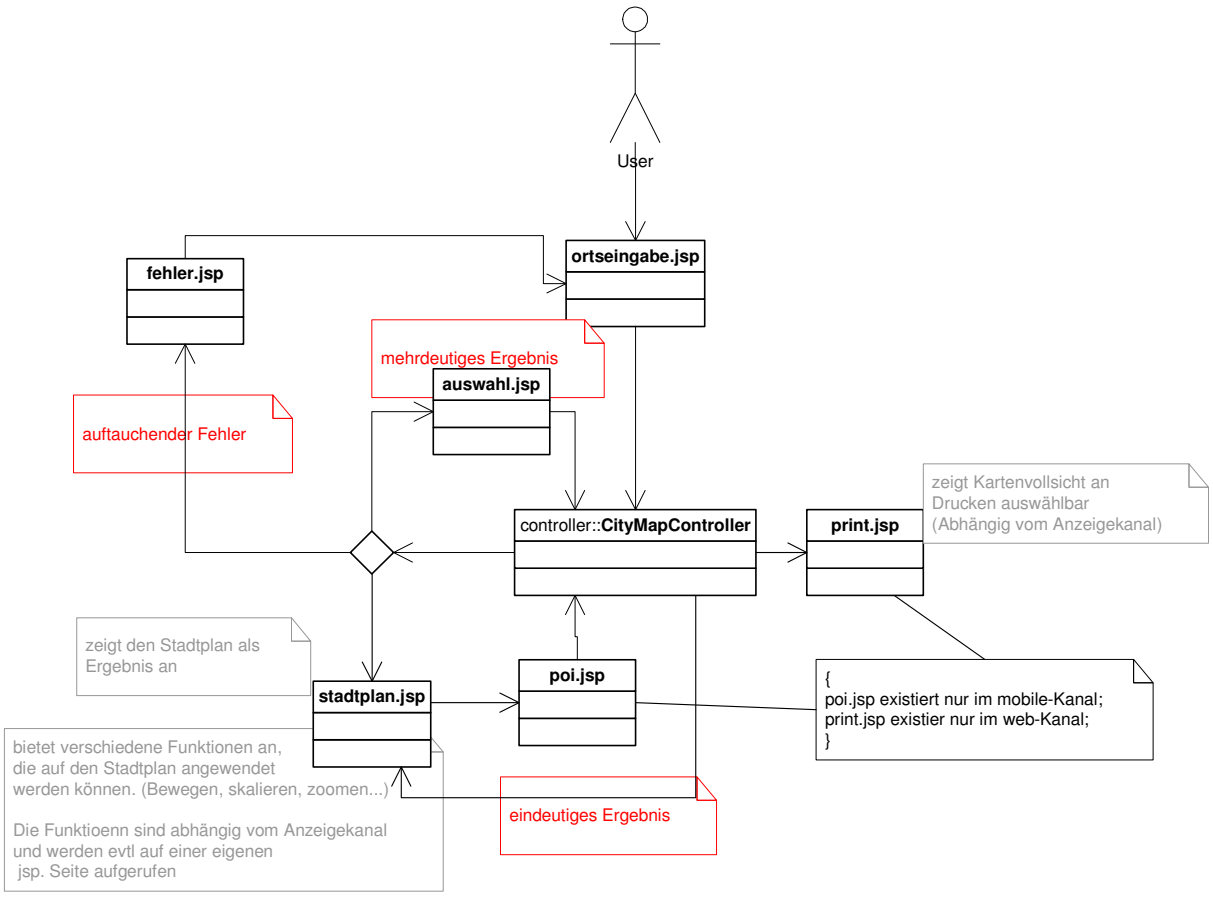
4.2.1 Funktionalitäten der Mobilen Applikation

- Adresse oder PLZ angebbar
- Auswählen 1 POI-Kategorie (POI: Point of Interests)
- Stadtplan anzeigen mit POI-Liste
- Zoom in, Zoom out
- Neue POI-Kategorie auswählbar

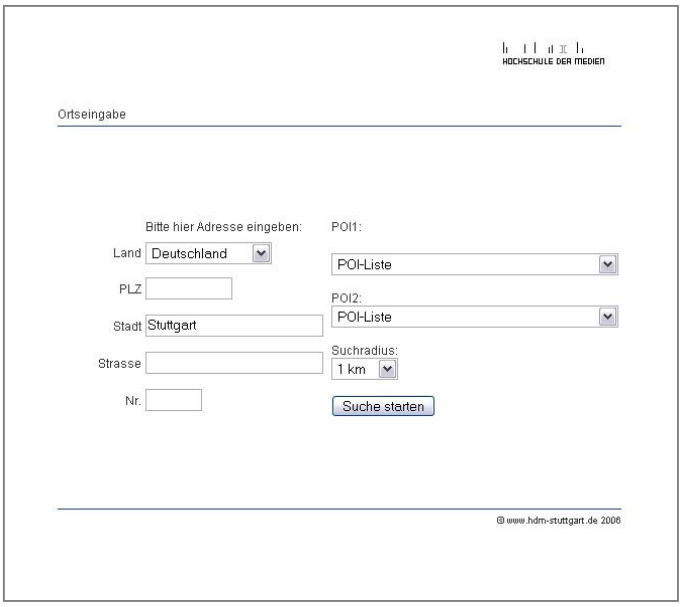
4.3 Unterschiede in den Funktionalitäten

Die Web-Applikation hat mehr Funktionalitäten als die Mobile Applikation. Dafür gibt es mehrere Gründe:
Zum einen ist die Datenübertragung im Web schneller, sodass mehr Daten auf einmal angefordert werden können, die dann in einer akzeptablen Zeit herunter geladen und angezeigt werden können. Aus diesem Grund wird die Ergebniskarte hier deutlich größer dargestellt.
Dann können zudem auch zwei POI-Kategorien ausgewählt werden und nicht nur eine wie bei der Mobilen Applikation. Das liegt daran, dass die Usability etwas schwierig und zu komplex würde, um die beiden POI-Kategorien auszuwählen, da dem User auf der mobilen Applikation nur einen Joystick zum Navigieren zur Verfügung steht. Und zum anderen gibt es natürlich den Grund der Datenübertragung.
Darüber hinaus kann man auf der Web-Applikation die POI-Kategorien direkt auf der Ergebnisseite neu auswählen, während man auf der Mobilen Applikation einen Button „POI" auswählen muss, um auf eine extra Seite zu kommen. Auch das wurde aus Usability – Gründen getan.
Es gibt für die Web-Applikation auch eine Druckansicht, auf der die Ergebniskarte mit den zugehörigen POI-Ergebnislisten ausgedruckt werden kann. Diese Option wird in einem neuen Fenster geöffnet. Diese Funktionalität ist in der Mobilen Applikation nicht vorhanden.
In der Web-Applikation kann man zudem auch die Karte in alle Himmelsrichtungen bewegen. Auch das kann man bei der Mobilen Applikation nicht, da allein für diese Funktionalität 8 Buttons durchgeklickt werden müssten, um auf den „Submit" Button oder zu anderen Funktionalitäten vorzudringen. Deshalb haben wir hier auf diese Funktionalität in der Mobilen Applikation verzichtet.

4.4 Kontrollfluß

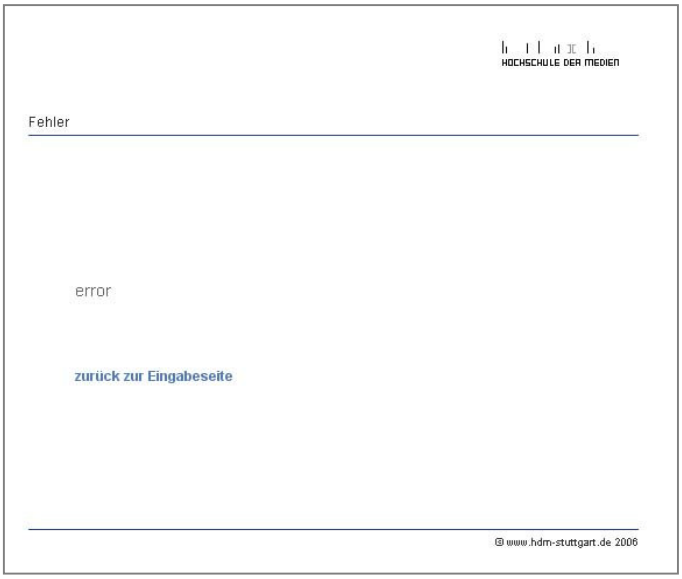


4.5 Screenshots der Web Applikation und Funktionsumfang



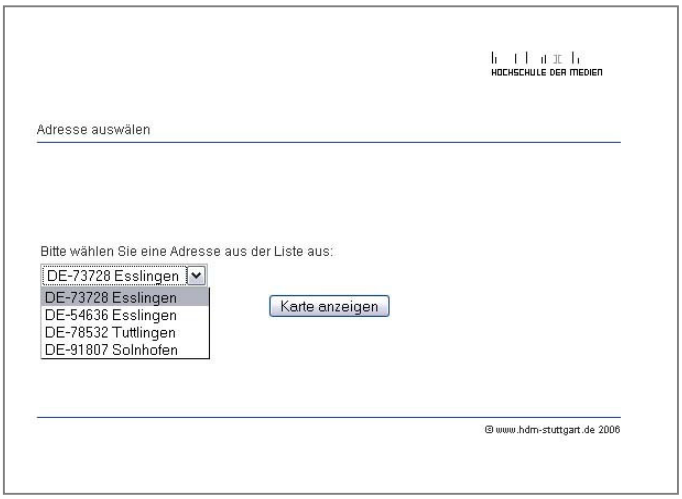
Ortseingabe:
Hier kann über die Adresseingabe ein Stadtplan angefordert werden. Zusätzlich können hier zwei POI Kategorien ausgewählt werden und zusätzlich der Suchradius ausgewählt werden.

Screnshot:
ortseingabe.jsp



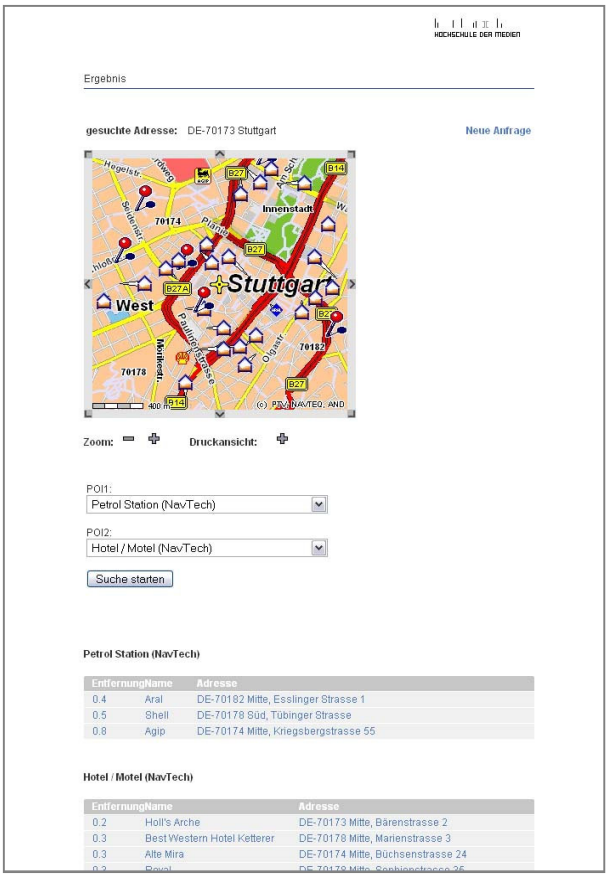
Error:
Im Fehlerfall wird man auf eine extra Errorseite weitergeleitet.

Screnshot:
error.jsp



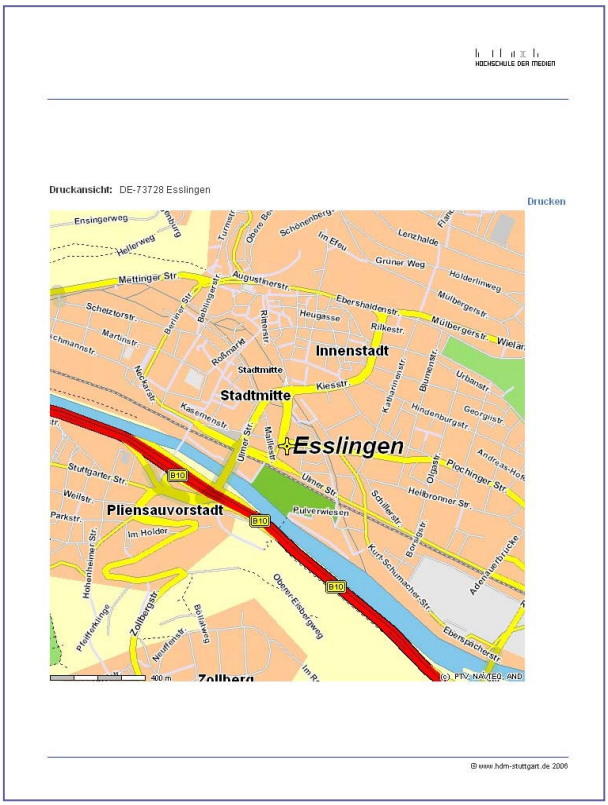
Adresse auswählen:
Falls eine Adresse nicht eindeutig ist, kann hier aus der Liste die gewünschte Adresse ausgewählt werden.

Screnshot:
auswahl.jsp



Kartenübersicht:
Hier wird der gewünschte Stadtplan mit evtl. POI-Kategorien angezeigt.
Hier kann man auch mit + und – in die Karte hinein oder heraus zoomen, sowie über „neue Anfrage" eine neue Adresse eingeben werden.
Zudem kann man hier die beiden POI-Kategorien neu auswählen.
Im Falle einer POI-Anzeige werden die POIs in einer Liste unterhalb angezeigt – sortiert nach der Entfernung und nach der Kategorie sortiert.
Es kann eine Druckansicht ausgewählt werden, die in einem neuen Fenster geöffnet wird.

Screnshot:
stadtplan.jsp



Druckansicht:
Hier wird der gewünschte Stadtplan mit evtl. POI-Kategorien in großer Druckansicht angezeigt.
Im Falle einer POI-Anzeige werden die POIs in einer Liste unterhalb angezeigt – sortiert nach der Entfernung und nach der Kategorie sortiert.
Mit „Drucken" lässt sich dieses Fenster ausdrucken.

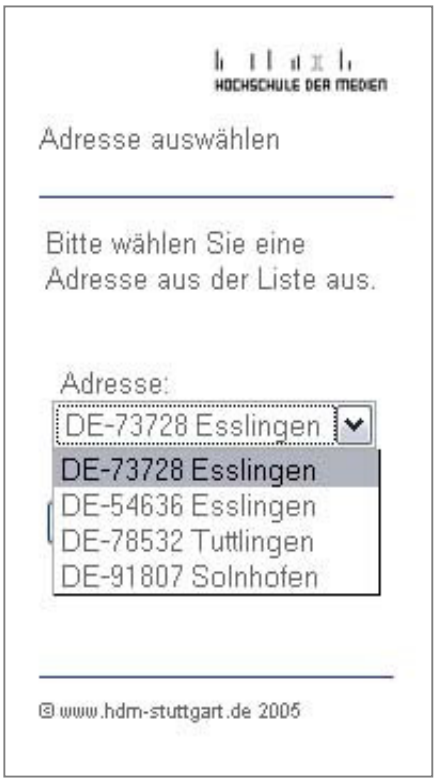
Screnshot:
druckansicht.jsp

4.6 Screenshots der Mobilen Applikation und Funktionsumfang

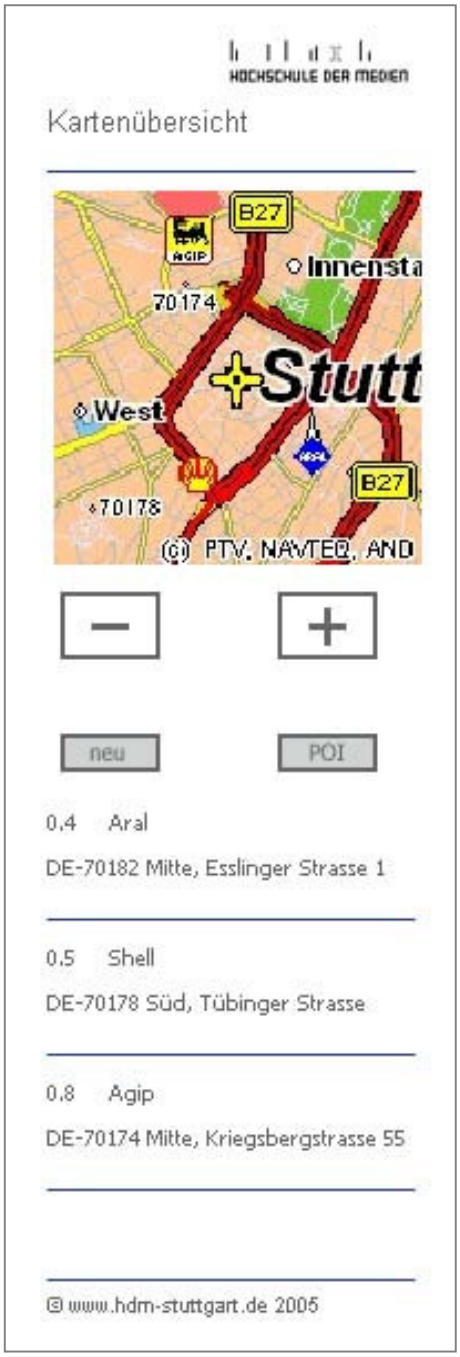
Ortseingabe:
Hier kann über die Adresseingabe ein Stadtplan angefordert werden.
Screenshhot: ortseingabe.jsp

POI auswählen:
Hier kann eine neue POI-Kategorie ausgewählt werden.
Screenshhot: poi.jsp

Error:
Im Fehlerfall wird man auf eine extra Errorseite weitergeleitet.
Screenshhot: error.jsp



Adresse auswählen:
Falls eine Adresse nicht eindeutig ist, kann hier aus der Liste die gewünschte Adresse ausgewählt werden.
Screenshhot: auswahl.jsp

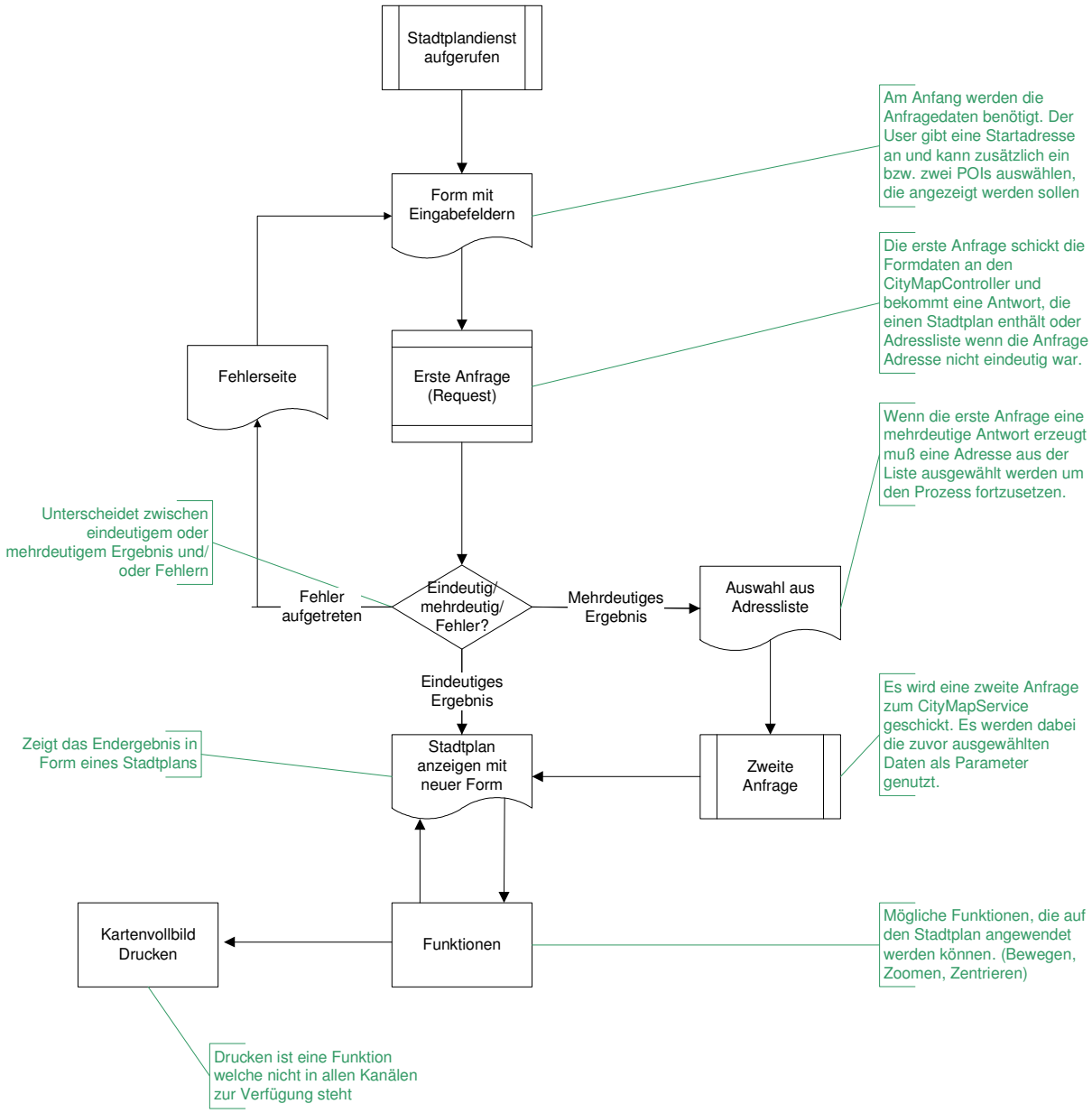


Kartenübersicht:
Hier wird der gewünschte Stadtplan mit evtl. POI-Kategorien angezeigt. Hier kann man auch mit + und – in die Karte hinein oder heraus zoomen, sowie über „neu" eine neue Adresse eingeben oder über POI eine neue POI-Kategorie auswählen.
Im Falle einer POI-Anzeige werden die POIs in einer Liste unterhalb angezeigt – sortiert nach der Entfernung.

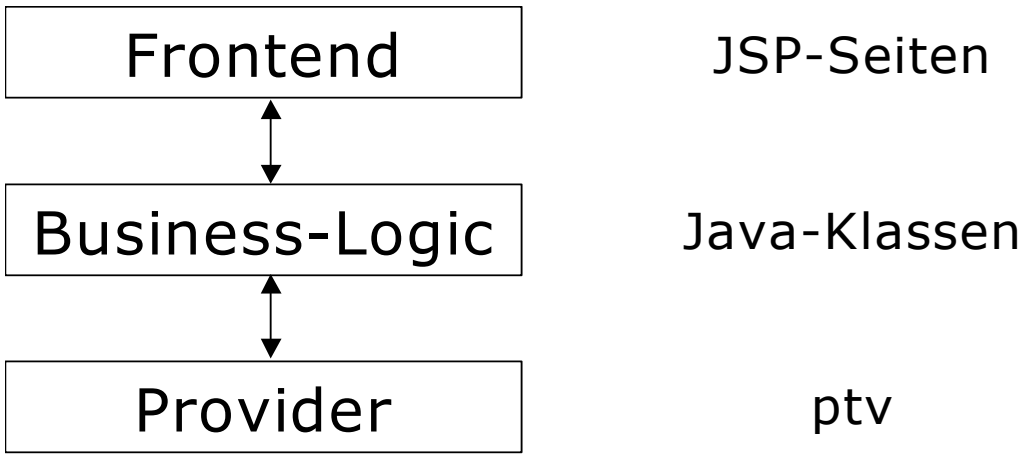
Screenshhot: stadtplan.jsp

5 Struktur / Aufbau

5.1 Flussdiagramm mobile appl / webfrontend

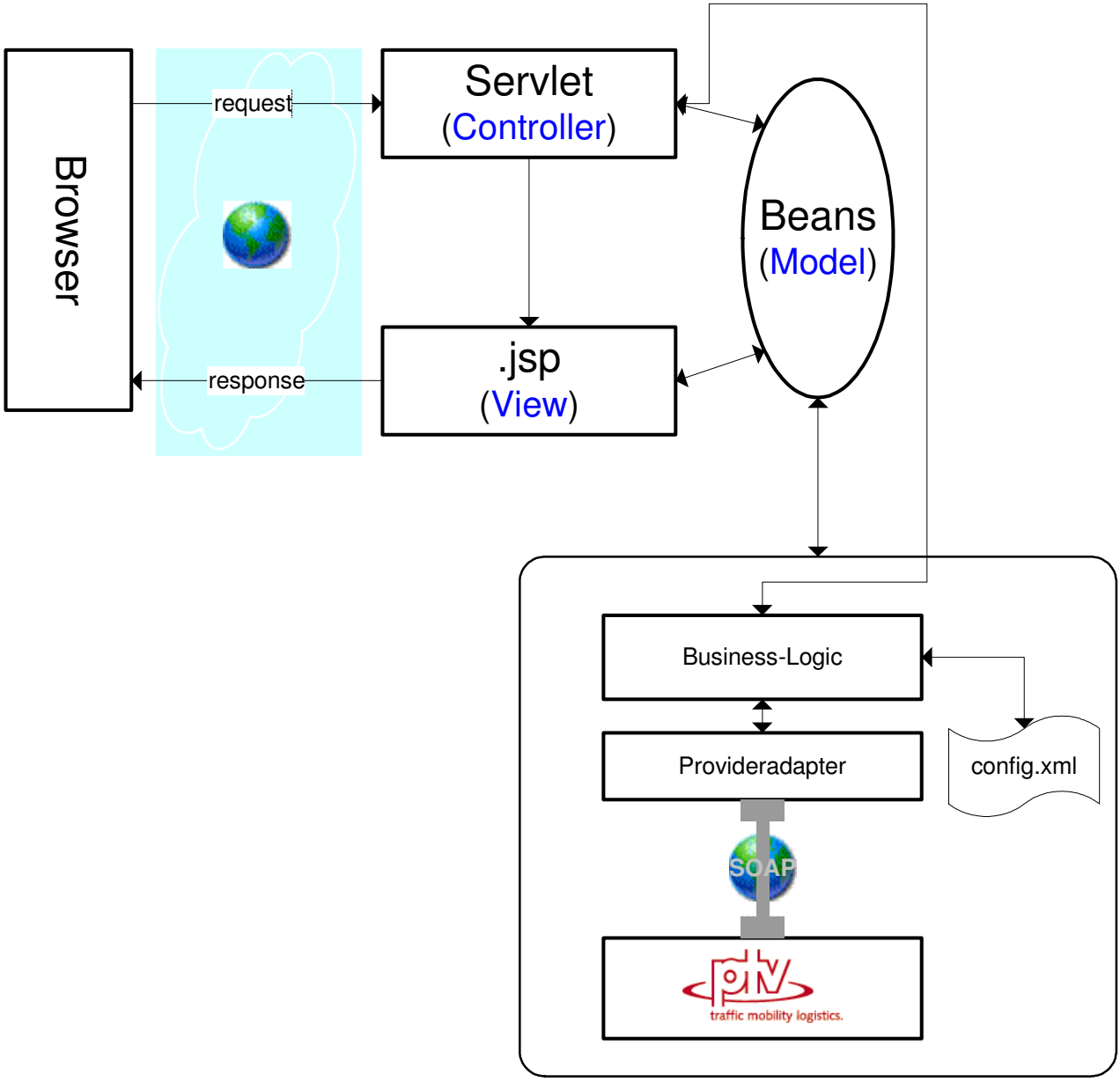


5.2 3-Schichtarchitektur



Die Trennung von Frontend und Business Logik war durch die Anforderung der verschiedenen Anzeigekanäle nahe liegend. Der Provider wird über Java WebServices mit der Business Logik verbunden.
Der Aufbau des Frontends wird in Kapitel 5.3 und 5.4 noch einmal detailliert betrachtet. Die Beschreibung der Business Logic ist in Kapitel 6 zu finden.

5.3 Systemaufbau / jsp model 2



Dieser Systemaufbau wurde gewählt, um das Modell View Controller Pattern innerhalb des Frontends umzusetzen.
Wie in Kapitel 4.4 zu sehen werden alle Abfragen von einer Komponente abgearbeitet. Durch den Einsatz dieses zentralen Controllers wurde die Umsetzung der Mehrkanalfähigkeit der Anwendung erheblich erleichtert.
Die Anbindung des Providers an die Businesslogik erfolgt über einen Adapter. Dies kapselt die spezifische Schnittstelle, sowie die Technologie zur Verbindung zwischen Provider und Applikation. Des Weiteren wird verhindert, dass innerhalb der Applikation Provider-Datentypen verwendet werden. Dies alles ermöglicht es, schnell und einfach auf Änderungen der Datenstruktur des Providers zu reagieren, die Zugriffstechnologie zu ändern oder sogar den gesamten Provider zu wechseln. Änderungen dieser Art wären nur sehr schwer durchführbar, würden alle providerspezifischen Daten und Aspekte nicht über einen Adapter gekapselt.

5.4 Aufruf des Controllers

Das Frontend setzt sich zusammen aus den jsp Seiten und dem zentralen Controller. Der Controller wird bei jeden Request aufgerufen, welcher die Be- und Verarbeitung notwendig macht. Einfache Links werden nicht über den Controller durchgeführt, sondern direkt mittels <a> Tag verlinkt.
Für die Eingabe und Übermittlung der Daten werden Web-Formulare eingesetzt. Im Attribut „action" im <form> Tag wird dabei festgelegt, an welches Element der Request mit den Daten gesendet werden soll.
Da im Frontend alle Requests an das zentrale ControllerServlet gestellt werden sieht die erste Zeile aller Formulare annähernd gleich aus.

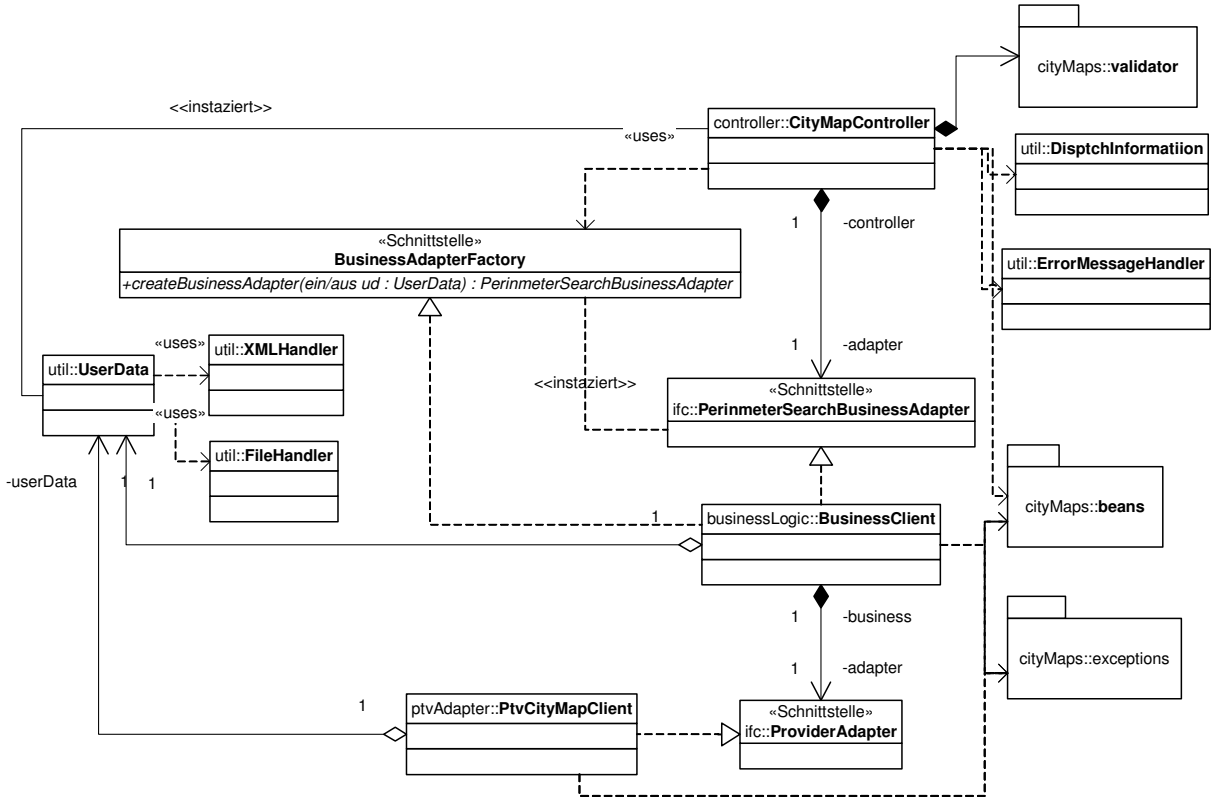
```
<form action="CityMapController" name= xxx method="GET">
```

Dabei ergibt sich folgendes Problem: Innerhalb des Controllers muss erkannt werden können, wie der eingegangene Request zu verarbeiten ist. Bei Anwendungen mit verteilten Controllelementen stellt sich dieses Problem nicht, da dort die Verarbeitung der Daten durch den Aufruf der entsprechenden Komponente festgelegt werden kann.
Aus diesem Grund enthält jedes Formular ein <input> - Feld mit dem Namen „ACTION". Durch dieses Request-Attribut können im Controller die Daten korrekt weiterverarbeitet werden.

```
<input type="hidden" name="action" value="GET_MAP_FROM_FORM"/>
```

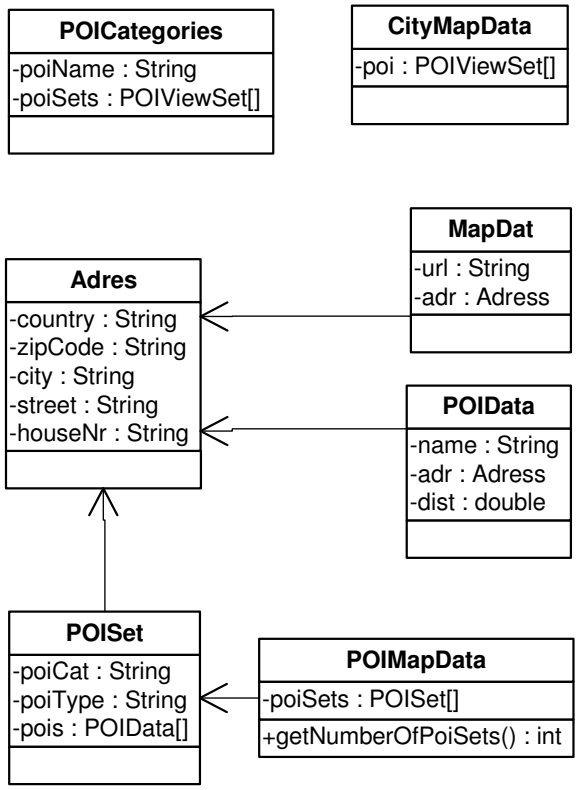
Einzige Ausnahme stellt der erste Aufruf des Controllers da. Dieser erfolgt direkt beim ersten Aufruf der URL innerhalb einer Session. Da dieser Aufruf nicht von einem Formular kommt, enthält er auch keinen Parameter „ACTION". Aus diesem Grund wird durch den Aufruf des Controllers ohne „ACTION" Parameter eine Initialisierung des Controllers angestoßen.

5.5 Strukturdiagramm der Business-Logik



5.6 Datenstruktur

Im Package "beans" sind folgende Klassen enthalten:



5.7 Beschreibung der verwendeten Klassen

Eine genaue Beschreibung der Klassen und ihrer Funktionsweise kann der JavaDoc des Projektes entnommen werden.

In dieser Dokumentation werden einige Besonderheiten konkrete Implementierungen erläutert.

5.7.1 Die Abstract Factory

Zur Instanzierung der Business Logik wird eine Abstract Factory verwendet. Diese wird eingesetzt, um die Trennung zwischen Frontend und Business Logik zu unterstützen. Die Factory wird verwendet durch den Aufruf der statischen Methode createBusinessAdapter(UserData ud). Die Factorymethode erwartet ein UserData-Objekt mit anzeigekanalspezifischen Informationen.
Da es sich bei der Factory um ein Interface handelt, stellt sich das Problem das Interface mit einer konkreten Implementierung zu versehen, ohne für den Benutzenden den Abstraktionsgrad der Abstract Factory zu verringern. Konkret bedeutet dies, dass alle Benutzer ausschließlich mit Referenzen auf das Interface BusinessAdapterFactory arbeiten. Um dies zu erreichen, besitzt das Interface das Attribut „FACTORY", welches die konkrete Implementierugsklasse festlegt und auf BusinessAdapterFactory castet. Um die Implementierung der Factory zu ändern, muss also an dieser stelle ein Objekt einer anderen Klasse zurückgegeben werden.

5.7.2 Das Package Validator

Das package Validator beinhaltet derzeit nur ein Interface „Validator". Diese Schnittstelle besitzt nur eine Methode validate(Objekt obj). Dies ist die Vorbereitung für eine Inputvalidierung innerhalb des Controllers, welche aus Zeitgründen nicht implementiert wurde.
Dahinter steckt folgende Idee:
Sollen Formulardaten validiert werden, muss für diese Daten eine Implementierungsklasse von Validator erstelllt werden. In der Methode validate() können die Regeln zur Überprüfung der Daten implementiert werden, diese liefert true oder false, entsprechend dem Überprüfungsergebnis zurück. Der Validator kann also durch das Hinzufügen einer Implementierung um eine Validierungsregel erweitert werden. Theoretisch wären auch Validatorimplementierungen denkbar, welche z.B. einen JavaScript Interpreter verwenden, und damit auch ermöglichen die Validierung anzupassen, ohne neu kompilieren zu müssen.

5.7.3 UserData

Diese Klasse beinhaltet alle für die BusinessLogik wichtigen Zusatzdaten. Dabei kann unterschieden werden zwischen allgemeinen Daten und kanalspezifischen Informationen. Sie wird in zwei Schritten initialisiert. Die Daten werden aus der Konfigurationsdatei config.xml ausgelesen. Daher kann eine Instanz der Klasse UserData als die Repräsentation der Daten aus der Konfigurationsdatei gesehen werden.
Die Klasse ist so implementiert, dass nur die kanalspezifischen Daten des gerade anzuzeigenden Ausgabekanals aus der config.xml geladen werden.

5.7.4 DispatchInformation

Diese Klasse wurde eingeführt, um den Kontrollfluss und die Dateinamen der jsp.Seiten des Frontends nicht direkt in den Sourcecode des Controllers verankern zu müssen, sondern um flexibel zu bleiben in ein Objekt auszulagern. Derzeitig werden diese Daten in HashMaps, welche im Constructor hart codiert gefüllt werden. Auch hier ist ein Mechanismus, ähnlich dem in UserData implementierten denkbar, die Daten in einer Datei vorzuhalten und bei Bedarf zu laden. Dadurch könnten sowohl der Kontrollfluss als auch die Dateinamen der jsp-Seiten jederzeit einfach angepasst werden.

5.7.5 ErrorMessageHandler

Um dem Endbenutzer verständliche Fehlermeldungen zu liefern, wurde der ErrorMessageHandler eingeführt. Er soll es ermöglichen, technische Fehlermeldungen in für den Benutzer verständliche Meldungen zu übersetzen. Auch diese Klasse ist bisher noch nicht korrekt implementiert, eine vollständige Implementierung, welche mit einer xml-Datei zur Festlegung der anzuzeigenden Fehlermeldungen realisiert wurde ist hier nahe liegend.

5.8 Die Datei config.xml

In dieser Konfigurationsdatei werden sowohl Kanalunabhängige Konfigurationsdaten als auch die kanalspezifischen Daten angegeben.
config.xml ist folgendermaßen aufgebaut:

```
<stadtplan>
  - <provider>
      <server>127.0.0.1</server>
      <user>xxx</user>
      <password>yyy</password>
    </provider>

  - <basicConfig>
      <defaultChannel>mobile</defaultChannel>
      <maxAdr>10</maxAdr>
      <geoDataSource>Europe</geoDataSource>
    </basicConfig>

  - <web>
      <mapHeight>1</mapHeight>
      <mapWidth>1</mapWidth>
      <deviceHeight>300</deviceHeight>
      <deviceWidth>300</deviceWidth>
      <scaleDevHeight>600</scaleDevHeight>
      <scaleDevWidth>600</scaleDevWidth>
      <zoomStep>25</zoomStep>
    </web>

  - <mobile>
      <mapHeight>1</mapHeight>
      <mapWidth>1</mapWidth>

                    ...
```

In der Gruppe „provider" werden alle Daten festgelegt, welche zum Verbindungsaufbau mit dem Provider benötigt werden.
Die darauf folgende „basicConfig" legt alle kanalunabhängigen Daten fest.
Darauf folgen die kanalspezifischen Daten. Für jeden Anzeigekanal wird eine neue Gruppe angelegt. Es können so beliebig viele Kanäle mit verschiedenen Ausgabeparametern definiert werden. An dieser Stelle wäre denkbar, die kanalspezifischen Gruppen noch um die Angaben zum Kontrollfluss oder den Dateinamen anzureichern.

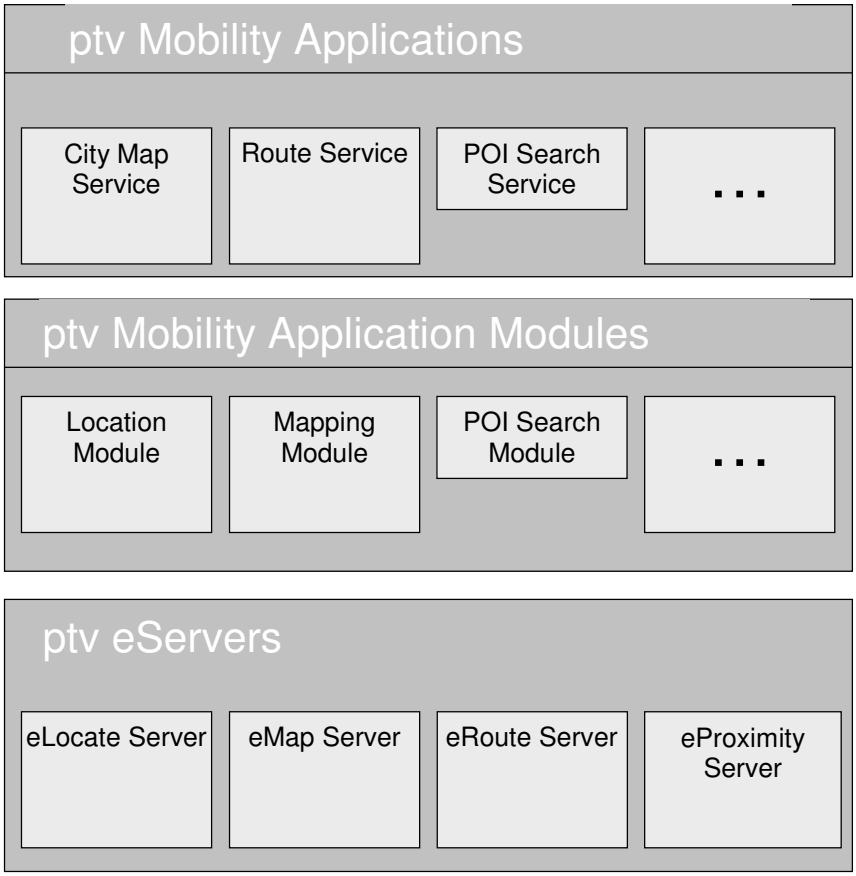
6 Schnittstelle zum Provider

6.1 Aufbau der ptv mobility Plattform

Die ptv Plattform ist in drei Schichten mit unterschiedlichen Abstraktionsgraden aufgebaut.

- **Applications**
  enthalten Funktionen um Standard Use Cases abzudecken. Sie bedienen sich dazu mehrerer Application Modules.
- **Application Modules**
  ermöglichen es, einfach auf die einzelnen Funktionen des Dienstes zuzugreifen. Sie nutzen evtl. mehrere eServer.
- **eServer**
  sind die abstraktesten und spezifischsten Komponenten.

Je nach Komplexität der eigenen Applikation, kann man sich dadurch wahlweise den fertig erstellten Applikationen bis hin zu den Basis Services der ptv direkt bedienen, um eine optimale Anbindung des Providers an die eigene Applikation zu gewährleisten.
Der Stadtplanservice verwendet die höchste Abstraktionsebene, die Mobility Applikations.



6.2 Axis Schnittstelle zum Provider

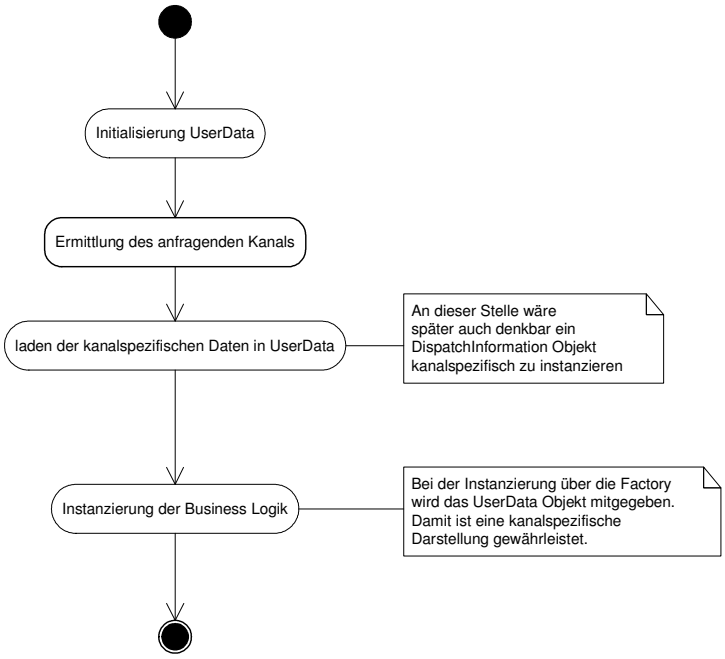
Die Anbindung des Providers erfolg mittels WebServices. Der zur Anbindung des Services an die eigene Applikation benötigte Axis-Proxy wird von dem Provider zur Verfügung gestellt. Er ist durch das jar-File „ws-client-2.2.jar" aus dem lib-Verzeichniss in die Applikation eingebunden. Die weiteren jar-Files mit Ausnahme der Dom-Implementierung „xercesImpl.jar" werden ebenfalls von dem ptv-Proxy verwendet.

7 Mehrkanalfähigkeit

Die Mehrkanalfähigkeit der Anwendung wurde durch die Kombination mehrerer Aspekte des Anwendungsdesigns erreicht. Alle beteiligten Aspekte werden noch einmal aufgezählt.

- Trennung von Frontend und Business-Logik
- Zentrales Controller Servlet
- Instanzierung der Business-Logik über eine Abstract Factory
- UserData Objekt, welches kanalspezifische Daten enthält
- Definierung der Daten sortiert nach Kanal in der config.xml
- Vorbereitung auch den Kontrollfluss und Dateinamen kanalspezifisch zu definieren

Die kanalspezifische Anzeige folge folgendem Ablauf:



Die Ermittlung des anfragenden Kanals erfolgt derzeit noch über einen Parameter, der ersten Anfrage, über ein Webformular.
Diese Information könnte durch das Auslesen der Browserkennung im ersten Request automatisiert werden, um eine automatisierte Browserweiche und Kanalerkennung zu realisieren. Die könnte in die init-Action des Controllers eingebaut werden.

8 XHTML MP und die Besonderheiten

8.1 Was ist XHTML MP?

XHTML MP (eXtensible HyperText Markup Language Mobile Profile) ist eine Auszeichnungssprache, die in WAP 2.0 definiert ist. WAP 2.0 ist die meist benutzte mobile Service Spezifikation, die vom WAP Forum erstellt wurde. Die WAP CSS Spezifikation (WAP Cascading Style Sheet or WCSS) ist auch in WAP 2.0 definiert. WAP CSS wird mit XHTML Mobile Profile zusammen benutzt.
XHTML Mobile Profile ist eine spezielle Form des XHTML, das wiederum eine strengere Version von HTML ist. XHTML Mobile Profile beruht auf XHTML Basics plus ein paar zusätzliche Elemente und Attribute und Attributen von der Vollversion von XHTML.
Das Ziel von XHTML Mobile Profile ist, die Technologien für das mobile Browsen im Internet und der fürs WWW zusammen zu bringen. Bevor es XHTML MP gab, benutzten die WAP Entwickler WML und WMLScript um WAP Seiten zu erstellen, während die Webentwickler HTML/XHTML und CSS Style Sheets benutzten um ihre Webseiten zu generieren
Mit der Veröffentlichung von XHTML MP näherten sich diese verschiedenen Auszeichnungssprachen einander an.
XHTML Mobile Profile und WAP CSS gaben den Entwicklern von drahtlosen Internetanwendungen die Möglichkeit, das Aussehen besser zu kontrollieren.

8.2 Vorteile von XHTML MP

Der größte Vorteil ist, dass dieselbe Technologie für beide Versionen der Internetseite benutzt werden kann. Man kann jeden Browser benutzen, um eine WAP 3.0 Anwendung während des Entwicklungsprozesses anzuzeigen.
Weitere Vorteile sind:
Wenn man sich mit HTML, CSS und XHTML auskennt, kann man quasi sofort WAP Seiten entwickeln, da die Unterschiede nicht so groß sind.
Wenn man sich noch nicht auskennt, muss man nur eine Technologie erlernen und nicht zwei verschiedene.
Es können dieselben Entwicklungswerkzeuge benutzt werden wie für normale HTML Seiten.
Gewöhnliche Webbrowser können zur Anzeige von WAP Seiten während des Entwicklungsprozesses benutzt werden. Man sollte allerdings die WAP Seiten mit einem Simulator zusätzlich testen wie dem Nokia Browser Simulator 4.0.

8.3 Besonderheiten von XHTML MP

Man muss ich im Klaren sein, dass natürlich ein Mobiles Endgeräte, weder die Farbauflösung, Displaygröße etc zur Verfügung hat wie ein normaler Browser. Deswegen sind natürlich auch die Attribute stark eingeschränkt.
Zudem ist die Übertragungsgeschwindigkeit momentan noch nicht allzu schnell, sodass auf möglichst viel verzichtet werden muss, was die Gestaltung der WAP Seite angeht.
Mit UMTS wird sich auch das wahrscheinlich etwas legen.

8.4 Beispiele von XHTML MP und die Unterschiede zu HTML

In den Tabellen wird oft die Entität „ “ verwendet. Das ist in XHTML MP nicht möglich. Hier sollten dann die Unicode Zeichen verwendet werden.

```
<tr>
   <td width="166"> </td>
</tr>
```

Auszug aus der XHTML MP – Datei der mobilen Applikation.

```
<tr>
   <td width="166"> </td>
</tr>
```

Auszug aus der HTML – Datei der Web Applikation.

Als weiteres Beispiel wäre zu nennen, dass die Seiten alle XHTML – konform sein müssen. Das bedeutet, dass alle Tags einen End-Tag besitzen müssen. Zum Beispiel das Tag
 ist so nicht benutzbar und führt zu einem nicht wohlgeformten Dokument, das nicht geparst werden kann.

```
<td class="schrift">
 Nr:<br />
.....
</td>
```

Auszug aus der XHTML MP – Datei der mobilen Applikation.

```
<td class="schrift">
 Nr:<br>
.....
</td>
```

Auszug aus der HTML – Datei der Web Applikation.

Zum anderen muss auch immer darauf geachtet werden, dass Attribute immer einen Wert besitzen. Das ist auch eine XHTML-Spezifikation, die aber strikt eingehalten werden muss.

```
<option selected="selected">
        POI-Liste
</option>
```

Auszug aus der XHTML MP – Datei der mobilen Applikation.

```
<option value='DE' selected >
        Deutschland
</option>
```

Auszug aus der HTML – Datei der Web Applikation.

9 Usability

9.1 Ergebnis der Usability-Untersuchung

Man kann sagen dass die Ladezeit etwas länger ist als beim Webfrontend, wobei wir mit deutlich längeren Ladezeiten gerechnet hätten. Wirklich schnell geht es wirklich mit UMTS
Positiv zu erwähnen ist außerdem, dass die Karte in dieser Größe noch gut erkennbar ist, obwohl noch nicht einmal die optimierte Version von den Kartendaten für mobile Endgeräte benutzt wurde
Der Dienst ist noch relativ teuer von den Kosten her, was aber auch an dem ungünstigen Prepaid-Tarif liegen könnte
Der Funktionsumfang muss geringer gehalten werden als auf dem Webfrontend, da die Usability mit dem Joystick bei vielen Links auf einer Seite wirklich schlecht ist
Deswegen ist auch nur 1 POI-Kategorie auf der mobilen Version auswählbar.Auf das Bewegen der Karte wurde ebenfalls aus diesen Gründen verzichtet

9.2 Folgerung / Fazit aus der Usability-Untersuchung

Die Bedienbarkeit ist bei wenigen Links akzeptabel.

Die Ladezeit ist erträglich, vor allem bei UMTS ist die Ladezeit überhaupt kein Problem mehr.
Wir haben keine Erklärung, warum bisher keine Realisierung bekannt ist – jedenfalls aus technischer Sicht.

10 Probleme

10.1 Aufgetretene Probleme

Die Verbindung zum ptv-Server ist teilweise nur über den HdM-Zugang möglich. Das bedeutete zu Anfang eine kleinere Schwierigkeit auf den Account zuzugreifen. Gegen Ende haben wir dann herausgefunden, dass man trotzdem direkt auf den ptv-Server zugreifen kann.
Die Browserweiche war in der Zeit nicht realisierbar. Hierbei war das Problem, dass man anhand des Request herausfinden muss, von welchem Endgerät die Anfrage gestellt wurde. Das hatte sich als zeitaufwendiger erwiesen als gedacht.
Dann traten noch öfters Störungen des ptv-Testservers auf wie Server busy – Meldungen. Da die Störungen meist nicht lang anhaltend waren, war dieses Problem nicht schwerwiegend für uns, hatte aber manchmal beim Testen ein wenig störend gewirkt.
Zudem war leider der Nokia Browser Simulator unbrauchbar. Die Darstellung der Seiten konnte leider überhaupt nicht als Anhaltspunkt genommen werden, da das Handy die Seiten dann doch ganz anders darstellte als der Simulator. Zudem haben wir es nicht geschafft, eine URL in diesem Simulator zu öffnen. Das bedeutete für uns – wir konnten nur statische Seiten testen.

10.2 Offene Punkte

Die Browserweiche konnten wir aus zeitlichen Gründen nicht mehr vollständig implementieren. Aber diese Funktionalität ist in der Planung berücksichtigt worden und ist auch in der Architektur abgebildet. Man muss im Prinzip nur noch eine Methode implementieren, die die tatsächliche Unterscheidung von Web und Mobil erledigt.

Man könnte diese Applikation natürlich in verschiedene Richtungen weiterentwickeln. Zum einen wäre sicher der Punkt „find me" interessant, der eine Karte vom aktuellen Standpunkt des Anfragenden liefert.

Zum anderen könnte man natürlich auf diesem Service aufbauend auch einen Routenplaner implementieren bzw. daraus entwickeln.

Es müsste natürlich viel mehr Testing betrieben werden, um die Funktionalität auf verschiedenen Mobilen Endgeräten zu testen und auch um Unterschiede in den verschiedenen Browsern herauszufinden. Die Simulatoren sind dazu nämlich momentan äußerst ungeeignet.