KaffeeKlatsch
Das Magazin rund um Software-Entwicklung

ISSN 1865-682X
01/2012

Mind the gap!

Plattformübergreifende mobile Entwicklung mit PhoneGap

von Werner Eberling

Mind the gap!

Plattformübergreifende mobile Entwicklung mit PhoneGap

von Werner Eberling

In keinem Bereich der Software-Entwicklung scheinen gerade so viele neue Projekte aus dem Boden zu sprießen, wie in der Welt der mobilen Endgeräte. Doch viele dieser Projekte haben mit der Problematik der heterogenen Systemlandschaft zu kämpfen. Dieser Artikel stellt mit *PhoneGap* eine Möglichkeit vor, systemübergreifend zu entwickeln, ohne auf den Zugriff auf Kamera, Kontaktliste oder GPS-Empfänger verzichten zu müssen.

Die Entwicklung für mobile Endgeräte (Smartphones, Tablet-PCs etc.) steht oft vor einem Problem, das der geneigte Java-Entwickler nur aus Erzählungen kennt: die Problematik zur Unterstützung verschiedener Betriebssysteme. Ob die zu entwickelnde Applikation unter *Android*, *iOS* oder *Windows Phone* laufen soll, entscheidet in der Regel grundlegend über die Technologie mit der sie realisiert wird. Soll eine Anwendung nun plattformübergreifend angeboten werden, bedeutet dies oft die Unterstützung verschiedener Programmiersprachen und damit de facto mehrere parallele und technisch unabhängige Projekte.

Hier wünscht man sich schnell die einheitliche Programmiersprache, die auf allen Systemen unterstützt wird. Bereits im letzten Jahr wurde an dieser Stelle der Einsatz von *HTML* und *JavaScript* (genauer gesagt: *jQuery*) zur Implementierung solcher Anwendungen vorgestellt [1]. Nachteil dieser Lösung: die Anwendung erscheint als Web-Seite nicht in der Applikations-Liste und der Zugriff auf Hardware-Funktionen des Gerätes (GPS-Empfänger, Kamera, etc.) sind, wenn überhaupt, nur proprietär möglich.

Unter anderem diese Probleme adressiert PhoneGap mit der Bereitstellung einer mobilen Applikationsplattform auf Basis von HTML 5 und JavaScript. Ursprünglich von *Nitobi* entwickelt, ist das unter *Apache Licence* stehende Open-Source-Projekt inzwischen bei *Adobe* untergekommen.[1] Es kapselt den Zugriff auf die nativen *Application Programming Interfaces* (API's) der unterstützten Betriebssysteme in JavaScript-Objekte und stellt so eine standardisierte Schnittstelle für die wichtigsten Hardware-Funktionen bereit (Unterstützung durch das jeweilige Endgerät vorausgesetzt).

Im Oktober 2011 wurde, unter dem Namen *Apache Callback*, sogar die Aufnahme als *Apache*-Projekt beantragt.

Von PhoneGap unterstützte Plattformen
Android
Blackberry
iOS
WebOS (HP)
Windows Phone
Samsung Bada
Symbian

Installation

Um mit PhoneGap zu entwickeln muss erst einmal die aktuelle Version von der Projektseite [2] heruntergeladen werden. Zum Entstehungszeitpunkt dieses Artikels war dies die Version *1.3.0*. Sie präsentiert sich nach dem Download als ZIP-Datei, die in einem Verzeichnis der Wahl entpackt werden kann. Ein Blick in den so entstandenen Verzeichnisbaum zeigt eine Liste von Ordnern, die die jeweiligen PhoneGap-Versionen für die verschiedenen mobilen Betriebssysteme enthalten.

Aber wieso verschiedene Versionen? Soll PhoneGap nicht systemübergreifende Entwicklung ermöglichen? Ja das ist das Ziel, aber dafür ist es notwendig eine Adapterschicht bereit zu stellen, die auf den jeweiligen spezifischen mobilen Systemen aufsetzt. Die Applikationsplattform, die so entsteht, ist dann immer die selbe, unabhängig davon ob sie auf Android, iOS oder Windows Phone betrieben wird.

[1] Die Beantwortung der sich hier aufdrängenden Frage über die Zusammenhänge zwischen dem steigenden Interesse von Adobe an HTML 5-Technologien und der zeitgleich eingestellten Unterstützung von *Adobe Flash* für mobile Endgeräte sei an dieser Stelle dem geneigten Leser selbst überlassen.

Für diesen Artikel soll Android als Beispiel für die Entwicklung mit PhoneGap herhalten. Installationsanleitungen für die anderen Plattformen sind im *Getting started*-Bereich der PhoneGap-Seite im Internet zu finden [3].

Voraussetzung für die Verwendung von PhoneGap mit Android ist das Android SDK [4] sowie die IDE der Wahl[2], die vorab mit dem passenden *Android-Plugin* versehen werden sollte [5][6]. Ist all dies installiert, wird eine „normale" Android-Anwendung erzeugt, die im Anschluss in wenigen Schritten zur PhoneGap-Anwendung umgebaut wird.

Als erstes müssen die von PhoneGap benötigten Dateien in die Anwendung kopiert werden. Diese befinden sich im Android-Verzeichnis der gerade ausgepackten PhoneGap-Installation. Hierzu werden im Wurzelverzeichnis der Android-Anwendung zwei neue Verzeichnisse angelegt:

- */libs*
  In dieses Verzeichnis wird das *phonegap.jar* kopiert.
- */assets/www*
  Hier liegen die eigentlichen Seiten der späteren PhoneGap-Anwendung. Zu Beginn wird hier die PhoneGap-JavaScript Bibliothek (*phonegap.js*) untergebracht.

Als letztes muss der *xml*-Ordner aus dem Android-Verzeichnis der PhoneGap-Installation als Ganzes in den bereits existierenden Ordner */res* des Android-Projektes kopiert werden.

Nachdem nun alle Dateien an ihrem Platz liegen, wird es Zeit PhoneGap in die Anwendung „einzuhängen". Hierzu wird die im Rahmen des Projektes angelegte *Activity* derart geändert, dass sie nicht mehr von *android.app.Activity* sondern von *com.phonegap.DroidGap* ableitet. Anschließend ist noch der Aufruf von *setContentView(...)* mit der Ladeanweisung für die HTML-Seite, die die eigentliche Anwendung darstellt, zu ersetzten:

```
public class MyGapActivity extends DroidGap {
  @Override
  public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    super.loadUrl("file:///android_asset/www/index.html");
  }
}
```

[2] Die PhoneGap-Gemeinde präferiert hier im Java Bereich *eclipse*, was sich deutlich in den Tutorials widerspiegelt.

Um Zugriff auf alle Hardware-Funktionen des mobilen Endgerätes zu haben, muss nun noch das Manifest der Anwendung geändert werden um alle benötigten Rechte zu erhalten.[3] Zusätzlich muss hier auch noch eine zusätzliche *Activity* registriert werden, die Teil der PhoneGap-Plattform ist:

```
<activity android:name="com.phonegap.DroidGap"
android:label="@string/app_name"
android:configChanges="orientation|
  keyboardHidden">
  <intent-filter> </intent-filter>
</activity>
```

Die erste Anwendung

Nun ist alles bereit für die erste PhoneGap-basierte Anwendung. Auch wenn hier klassisch eigentlich ein *Hello World* kommen müsste, soll bereits das erste Beispiel einen ersten kleinen Eindruck von den Möglichkeiten zur Kommunikation mit dem Endgerät geben.

```
<html>
  <head>
    <title>Hello Device</title>
    <script type="text/javascript" charset="utf-8"
    src="phonegap-1.3.0.js"></script>
    <script type="text/javascript" charset="utf-8">
    document.addEventListener("deviceready", devInfo, false);
    function devInfo() {
      var element = document.getElementById('info');
      element.innerHTML = 'Name: ' + device.name +
      ' (UUID:' + device.uuid + ')<br />' + 'Platform: ' +
      device.platform + ' ' + device.version;
    }
    </script>
  </head>
  <body>
    <p id="info">Lese Daten aus...</p>
  </body>
</html>
```

Das in diesem Beispiel verwendete *device*-Objekt ist Teil der PhoneGap-API [7] und bietet die Möglichkeit einfache Informationen des mobilen Endgerätes auszulesen. Der verwendete Event *deviceready* wird gefeuert sobald PhoneGap fertig initialisiert ist. Es stellt sozusagen die PhoneGap-spezifische Version des aus der JavaScript-Entwicklung bekannten *document.ready*-Events dar.

[3] Auf die Darstellung des entsprechenden XML-*Snippets* wurde hier aus Gründen der Übersichtlichkeit verzichtet. Für die benötigten XML-Einträge sei an dieser Stelle auf das PhoneGap-Tutorial im Internet verwiesen [3].

Wird die beschriebene Seite als *index.html* in den Ordner */assets/www* des Android-Projektes gespeichert, so ist die Anwendung bereit zum Deployment. Dieses findet mit den „normalen“ Bordmitteln des gewählten Android-Plugins statt und kann sowohl in den lokalen *Android-Simulator* als auch auf ein reales Endgerät erfolgen. Nach dem Start der Anwendung erscheint die folgende Ausgabe (die Werte entsprechen einem *HTC Incredible S*):

```
Name: htc_vivo (UUID:758fdd38c0187a14)
Platform: Android 2.3.5
```

Weitere Objekte der PhoneGap-API

Das Auslesen von Geräteinformationen ist nur ein erster kleiner Schritt in die Welt der Möglichkeiten, die PhoneGap bietet. Die API bietet Objekte zum Zugriff auf alle wichtigen Funktionen des mobilen Endgerätes. Egal ob Kontaktliste, Kamera, Kompass, Beschleunigungsmesser oder GPS-Modul, diese und noch weitere Funktionen sind über einfache JavaScript-Funktionen steuer- bzw. abfragbar. Das nachfolgende *JavaScript-Snippet* nutzt diese API um die aktuelle GPS-Position und den vom Kompass gemeldeten Steuerkurs auszulesen und darzustellen.

```
//...
function posAndHead() {
  navigator.geolocation.getCurrentPosition(
    gpsSuccess, onError
  );
  navigator.compass.getCurrentHeading(
    headSuccess, onError
  );
}

function gpsSuccess(position) {
  var element = document.getElementById('pos');
  element.innerHTML = 'Laengengrad: ' +
  position.coords.longitude +'<br />Breitengrad: ' +
  position.coords.latitude ;
}

function headSuccess(heading) {
  var element = document.getElementById('head');
  element.innerHTML = 'Kurs ' +
  heading.magneticHeading;
}
//...
```

Ruf mich an!

Und wie sieht es mit dem Tätigen von Telefonanrufen aus? Hier zahlt es sich aus, dass PhoneGap auf HTML 5 aufbaut. Der aktuelle HTML-Standard kennt den Datentyp einer Telefonnummer und ermöglicht beispielsweise mit dem Protokoll *tel* die Verwendung solcher Nummern als Hypertext-Links.

```
<a href="tel:+49-9131-89030">
MATHEMA anrufen...
</a>
```

Entwicklung für verschiedene Plattformen

JavaScript und HTML 5 bilden die Sprachmittel um mit PhoneGap Anwendungen für mobile Geräte zu entwickeln. Obwohl der eigentliche Applikationscode damit nicht mehr gerätespezifisch ist, bleibt die spezifische Lösung zur Integration von PhoneGap in das jeweilige Betriebssystem. Somit reduziert sich zwar der systemspezifische Anteil, er ist aber immer noch vorhanden. Wie gelingt es nun mit diesem Ansatz eine Applikation für verschiedene Plattformen (z. B. Android und iOS) basierend auf einem einzigen Entwicklungsprojekt zu realisieren?

Hierzu existieren verschiedene Ansätze. Wer sich für die „do-it-yourself“-Variante entscheidet, findet im Internet eine Reihe an Tutorien, die beschreiben, wie systemspezifische Anteile in eigene Projekte ausgelagert werden können, die mit Mitteln gängiger Versionskontrollsysteme (z. B. *svn:external*) mit dem eigentlichen, systemübergreifenden Applikationscode-Projekt „verlinked“ werden. Wem dies zu viel „Bastelarbeit“ ist, für den bietet PhoneGap ein *Cloud*-basiertes *Buildsystem* [8], das nach Übergabe des PhoneGap-basierten Anwendungscodes (bestehend aus HTML, CSS und JavaScript) die fertige Applikation zurückliefert.[4]

Resümee

Als geräteübergreifende Entwicklungsplattform macht PhoneGap auf den ersten Blick einen recht guten Eindruck. Die Notwendigkeit zur Auseinandersetzung mit den Spezifika der Zielsysteme wird auf ein Minimum reduziert. Gleichzeitig bleibt der Zugriff auf die wichtigsten Hardware-Funktionen erhalten. Natürlich ist auch hier nicht alles Gold, was auf den ersten Blick glänzt. Eine der wichtigsten Sektionen in der API-Dokumentation der einzelnen Objekte ist jeweils der Abschnitt über die systemspezifischen Einschränkungen. Hier muss bei der

[4] Auf eine Diskussion bzgl. Datenschutz und Urheberrechte wird an dieser Stelle verzichtet. Der interessierte Leser sei auf die Lizenzbestimmungen des PhoneGap-Buildsystems verwiesen.

Entwicklung eben doch wieder Rücksicht auf die Zielplattform genommen werden, um nicht Funktionen zu verwenden die nur auf zwei der drei Zielsysteme vorhanden sind.
Ein weiteres mögliches Problem ist die benötigte HTML 5-Unterstützung. Hier spielt allerdings die Zeit für PhoneGap, da zu erwarten ist, das die Unterstützung für den neuen HTML-Standard in Zukunft immer besser werden wird.

Ausblick

Dennoch ist PhoneGap mit Sicherheit einen Versuch wert, wenn es darum geht Systemspezifika bei der mobilen Entwicklung zu vermeiden. Besonders interessant dürfte hierbei die Kombination aus PhoneGap und *jQuery Mobile* [9] sein. Neben der Möglichkeit zum einfachen Zugriff auf Hardware-Funktionen mit PhoneGap bietet jQuery Mobile eine einfache Basis zur Entwicklung von mobilen User Interfaces.[5]

Also: Mut zur Lücke! Sie ist mit den richtigen Hilfsmitteln deutlich kleiner, als es auf den ersten Blick erscheinen mag.

Ressourcen

[1] Eberling, Werner *Geh weg, bleib da! - Anbindung mobiler Clients mit jQuery und REST*, KaffeeKlatsch 05/2011, Bookware, Erlangen 2011

[2] PhoneGap
http://phonegap.com

[3] PhoneGap *Getting started*
http://phonegap.com/start

[4] ANDROID developers *SDK*
http://developer.android.com/sdk/index.html

[5] NetBeans *NBAndroid*
http://plugins.netbeans.org/plugin/19545/nbandroid

[6] ANDROID developers *ADT Plugin für eclipse*
http://developer.android.com/sdk/eclipse-adt.html

[7] PhoneGap Dokomentation *API Reference*
http://docs.phonegap.com/en/1.3.0/index.html

[8] PhoneGap:Build
https://build.phonegap.com

[9] jQuery *mobile framework*
http://jquerymobile.com

[10] MDS *AppLaud Plugin*
http://www.mobiledevelopersolutions.com/home

Kurzbiographie

Werner Eberling ist Principal Consultant und Technical Lead bei der MATHEMA Software GmbH in Erlangen. Seit 1999 beschäftigt er sich mit verteilten Systemen, sein Fokus liegt dabei auf CORBA und der JEE. Neben seiner Projekttätigkeit hält er Technologie-Trainings und ist als Sprecher auf nationalen und internationalen Konferenzen anzutreffen. Sein aktuelles Steckenpferd ist die Anwendungsentwicklung für mobile Endgeräte bzw. deren Anbindung an Enterprise Systeme. Werner Eberling ist unter anderem Autor des Buches *Enterprise JavaBeans 3.1, Das EJB-Praxisbuch für Ein- und Umsteiger*, erschienen im Hanser Verlag.

[5] Das Duo hat es inzwischen sogar schon zu einem eigenen *eclipse*-Plugin gebracht [10].