Hybride mobile Applikationen mit ADF und PhoneGap

Christof Kaller
MT AG
Ratingen

Schlüsselworte
Java, ADF, PhoneGap, Android, CSS, Javascript

Schwerpunkt	Java, ADF, PhoneGap, Android, CSS, Javascript
Vortragstyp	Minilesson
Ziellevel	Fortgeschrittene
enthält Demo	Ja
Status	angenommen
Vortragsmanuskript	Ja
Präsentation	Ja
auf der DOAG	Mi, 16.11., 10 Uhr, Raum Prag
Jahr	2011

Einleitung

Will man heute den Softwaremarkt für mobile Endgeräte abdecken, muss man eine Vielzahl verschiedener Betriebssysteme und Endgeräte unterstützen. Hierdurch entsteht erhöhter Personalbedarf bzw. Bedarf an gebündeltem Know-how: Für jede Plattform müssen qualifizierte Entwickler, die ein Softwareprodukt entwickeln und auf unterschiedliche Plattformen portieren können, zur Verfügung stehen. Dies erhöht sowohl den Entwicklungs- als auch den Wartungsaufwand. Aus diesem Grund entstehen momentan mehrere Frameworks, die von konkreten Geräten und Betriebssystemen abstrahieren. Diese Frameworks bieten eine Schicht, gegen die man entwickeln kann und die auf allen wichtigen Smartphones vorhanden ist.

Die meisten mobilen Geräte wie z.B. Smartphones haben nur eine Gemeinsamkeit: den Zugang zum Internet. Eine Web-Applikation vom Browser des mobilen Gerätes aus aufzurufen, ist also der offensichtlichste Weg, eine ADF-Anwendung auf einem Smartphone zu nutzen. Jedoch haben Web-Applikationen in der Regel keinen Zugriff auf native Funktionen wie z.B. Kontakte oder den Speicher des mobilen Geräts.

Durch die Nutzung von Frameworks wie PhoneGap bieten sich dem Entwickler völlig neue Möglichkeiten. Ein aufgebohrter, um zusätzliche Funktionen erweiterter Browser bietet Entwicklern die Chance, aus ADF-Anwendungen heraus über JavaScript auf diese nativen Funktionen zuzugreifen und auch Daten auf dem mobilen Gerät zu manipulieren.

In einer Live Demo wird gezeigt, wie man mit einfachen ADF und JavaScript-Befehlen verschiedene Funktionen aufrufen und nutzen kann, um eine Anwendung zu erstellen, die die Möglichkeiten moderner mobiler Geräte voll ausschöpfen kann.

Vorbereitung

Smartphone mit Android als Betriebssystem

Eclipse mit benötigten Android SDK

PhoneGap Projekt Sourcen

JDeveloper mit Mobile Extension & Server

-> Computer mit 4 GB RAM

Motivation und Ziele

Ziel dieses Vortrags ist es dem Entwickler eine Möglichkeit zu zeigen wie man mit einigen wenigen Handgriffen ADF Mobile Webapplikation wie native Anwendungen aussehen und bedienbar machen kann. Und dabei auch noch native Funktionen von Smartphones nutzen kann. Sodas das Ziel: bei deutlich geringerem Entwicklungsaufwand ein vielfaches Bereitstellen ermöglicht werden und so größtmögliche Effizienz erzielt werden kann.

Folgende nativen Funktionen und Features sollen dabei näher betrachtet werden:

- Installation auf den Gerät (ein Button für den Desktop/Homescreen)
- Styling durch CSS (individuell für jedes Gerät)
- Aufruf von nativen Funktionen (Abfragen von GPS Koordinaten/ erstellen eines Fotos)

Aufbau und Vorgehensweise

Der Vortrag besteht aus zwei Teilen. Der erste Teil wird dabei in Form einer Präsentation die Theorie erläutern und wesentliche Konzepte wie Architektur und Erweiterbarkeit vorstellen. Schrittweise wird dabei der Weg zur fertigen App aufgezeigt.
Anschließend folgt im zweiten Teil eine Live Demo bei der anhand von Beispielen gezeigt werden soll wie die Theorie in der Praxis umgesetzt werden kann.

Technischer Hintergrund –Vorwissen

ADF Mobile. Mit ADF Mobile hat Oracle sein Portfolie um eine Möglichkeit für Entwickler, Oracle ADF Applikationen, auf mobile Geräte zu portieren erweitert. Die Extension steht zum Download bereit und kann über ein Update im JDeveloper oder Manuell installiert werden. Nach der Installation kann ein Entwickler speziell auf mobile Geräte zugeschnittene „Apps" entwickeln. Dabei stehen ihm die üblichen bekannten Wizzards zur Verfügung. Man kann vorhandene Libraries aus anderen Oracle ADF Anwendungen einbinden und nutzen. Wie bei ADF sonst auch ist vieles deklarativ. Man kann Workflows erstellen und hat die Möglichkeit die Anwendung in verschieden Auflösungen zu testen. Allerdings kann man bisher nur trinidad Komponenten verwenden.

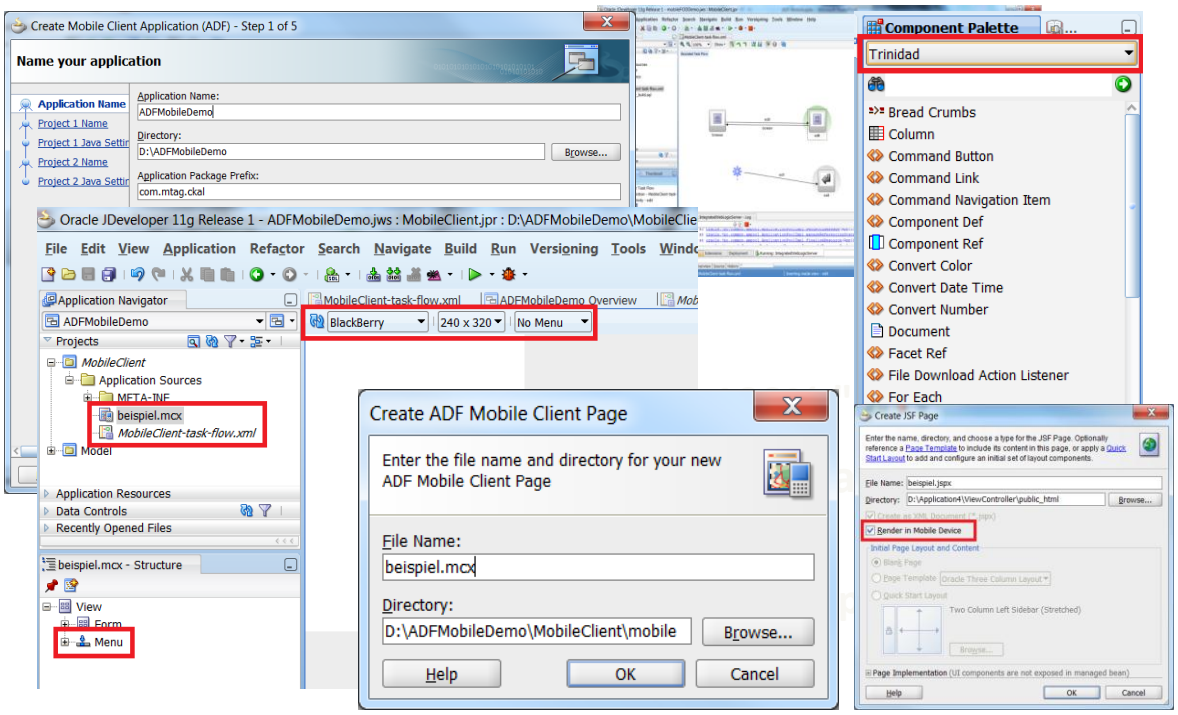

Abbildung 1: Übersicht Wizzards und Komponenten

Oracle ADF Mobile gibt es zurzeit noch[1] in zwei Varianten: ADF Mobile Client und ADF Mobile Web. Beide bieten verschiedene Vorteile haben auch Nachteile.

[11] Auf der Open World soll eine neue Version vorgestellt werden. vgl. ADF Entwickler Blog

Mobile Browser

- Real- Time Data
- Basisintegration von nativen Funktionen
- Breite Unterstützung mobiler Geräte
- Sehr schnelle Entwicklung - go-alive today

Mobile Client

- Offline Verfügbarkeit von Daten
- Breite Unterstützung nativer Funktionen (GPS, Kontakte,…)
- Konsistente Client Performance
- Eingeschränkte Unterstützung mobiler Geräte

Um die Nachteile der einen Variante wie etwa: eingeschränkte Unterstützung mobiler Geräte, auszumerzen und die Vorteile der anderen, breite Unterstützung mobiler Geräte, nutzen zu können muss man sich eines anderen Hilfsmittels bedienen.

Android das Betriebssystem von google auf dem die Live Demo dieser Präsentation basiert, basiert auf Java „ist aber keins". Was heißt das? Es gibt Klassen, Methoden, Objekte... Die Syntax ist gleich und die Semantik ist der von Java sehr ähnlich. Ein geübter Java Entwickler wird sich schnell einarbeiten können. Inzwischen werden die aus Java bekannten Java-„Core" Klassen vom Compiler unterstützt und werden damit in der eigenen Dalvik VM lauffähig. Es gibt ein paar zusätzliche Android spezifische APIs und Methoden (z.B. die aus Java bekannte main Methode heißt hier: onCreate). Online findet man zahlreiche Tutorials welche dies ausführlich beschreiben.

Entwickler können kostenlos das Android SDK[2] installieren, welches sich hervorragend in die empfohlene Entwicklungsumgebung Eclipse[3] integrieren lässt und dann zahlreiche Erweiterungen bietet. Beispiele für solche Erweiterungen:

- Anlegen eines neues Android Projekts,
- Nutzung des Dalvik Debug Monitor Service zum grafischen Debuggen,
- Konvertierung eines Projektes nach Android,
- Nutzung des Android Virtual Device Manager[4],
- Signieren und Installieren von Projekten und
- Nutzung des komfortablen Kommandozeilen - Ebenen Tools adb[5].

Lizenzkosten fallen nur dann an wenn man Apps im Android Market Place veröffentlichen will.

PhoneGap. Da es keinen Standard wie JavaME für alle Plattformen gibt und JavaME auch nicht von allen Plattformen unterstützt wird, benötigen wir ein alternatives Framework. Dieses Framework soll übergreifend für alle Plattformen nutzbar sein und sicherstellen, dass das vorhandene Know-how der Entwickler auch weiterhin eingesetzt sowie deren besondere Erfahrungen bei der Entwicklung ADF basierter Webanwendungen sinnvoll weitergenutzt werden kann.

[2] Software Development Kit (SDK) (engl.): ein SDK ist eine Sammlung von Tools, APIs und Dokumentionen zur Entwicklung.
[3] Open Source Entwicklungsumgebung mit vielen Erweiterungsmöglichkeiten.
[4] Tool zum Starten und Verwalten von Virtuellen Maschinen.
[5] Android Debug Bridge (adb): zum Loggen und Debbugen.

Plattformübergreifende Frameworks wie PhoneGap vereinen die Gemeinsamkeiten aller Plattformen, stellen zusätzliche Funktionen zur Verfügung und funktionieren alle nach einem ähnlichen Prinzip: sie bilden einen Container pro Plattform, und bieten eine, für alle Plattformen gleiche, Schnittstelle, über welche auf die nativen Funktionen zugegriffen werden kann. Sie sind somit ein Werkzeug zum Programmieren von Anwendersoftware für verschiedene Plattformen und sind für Cross-Programmierungsaufgaben gut geeignet.

PhoneGap ist ein solches Framework. PhoneGap ist ein Open Source Projekt mit Ursprung in San Francisco. Hat eine sehr aktive Community und bietet gute Tutorials.

PhoneGap hebt sich von den anderen Frameworks vor allem dadurch ab, dass sich die Entwickler von PhoneGap grundsätzlich an Standards orientieren und von Anfang an möglichst viele Plattformen unterstützen. Wenn man den PhoneGap Entwicklern bei Twitter folgt, so wird die enge Zusammenarbeit mit der WAC[6] und die konsequente Umsetzung von Standards transparent. Das nutzbringende Vorgehen wird besonders dadurch sichtbar, dass beispielsweise die von PhoneGap genutzte Funktion zur Standortabfrage inzwischen auch von einigen Desktopbrowsern, wie z.B. Opera, unterstützt wird. Dies führt dazu, dass der Aufruf der Beispiel PhoneGap index.html Datei aus dem PhoneGap Demoprogramm mit dem Browser den aktuellen Standort ermittelt. Die Ermittlung des Standortes erfolgt hier nicht per GPS sondern geschieht auf Basis der IP Adresse.

Nach dem Import des Android Projekt Quellcodes in Eclipse sieht man, dass eine Index.html im Ordner „www“ unter „assets“ wieder zu finden ist. Hinzugekommen ist außerdem die PhoneGap.js Datei, welche die entsprechend (vom Build- Skript) generierte JS-Schnittstelle zur Verfügung stellt. Die phonegap.jar Datei entspricht dem Container welche die Schnittstelle mit dem nativen Code verknüpft (ausführliche Beschreibung der js Datei fehlt aus Zeitgründen). Nach dem Kompilieren wäre die Anwendung anschließend im Emulator lauffähig.

Eine PhoneGap Applikation besteht im Wesentlichen aus drei Komponenten:

- der Webapplikation (incl. index.html),
- der Schnittstelle (PhoneGap.js),
- dem nativen PhoneGap Skelet (DroidGap bzw. PhonGap.jar).

Dabei kann man zwischen einen Plattformabhängigen und unabhängigen Teil unterscheiden.

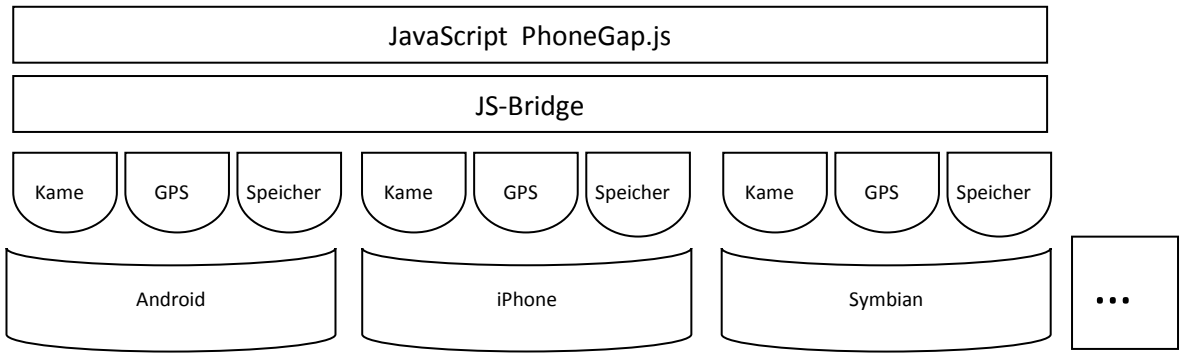

Abbildung 2: drei Plattformen drei Betriebssysteme

[6] Wholesale Application Community & auch JIL

Bevor im Folgenden die einzelnen Komponenten näher erläutert werden, wird zunächst das Zusammenspiel der verschiedenen Schichten beschrieben. Das ist wichtig für das Verständnis der asynchronen Verarbeitung der Daten.

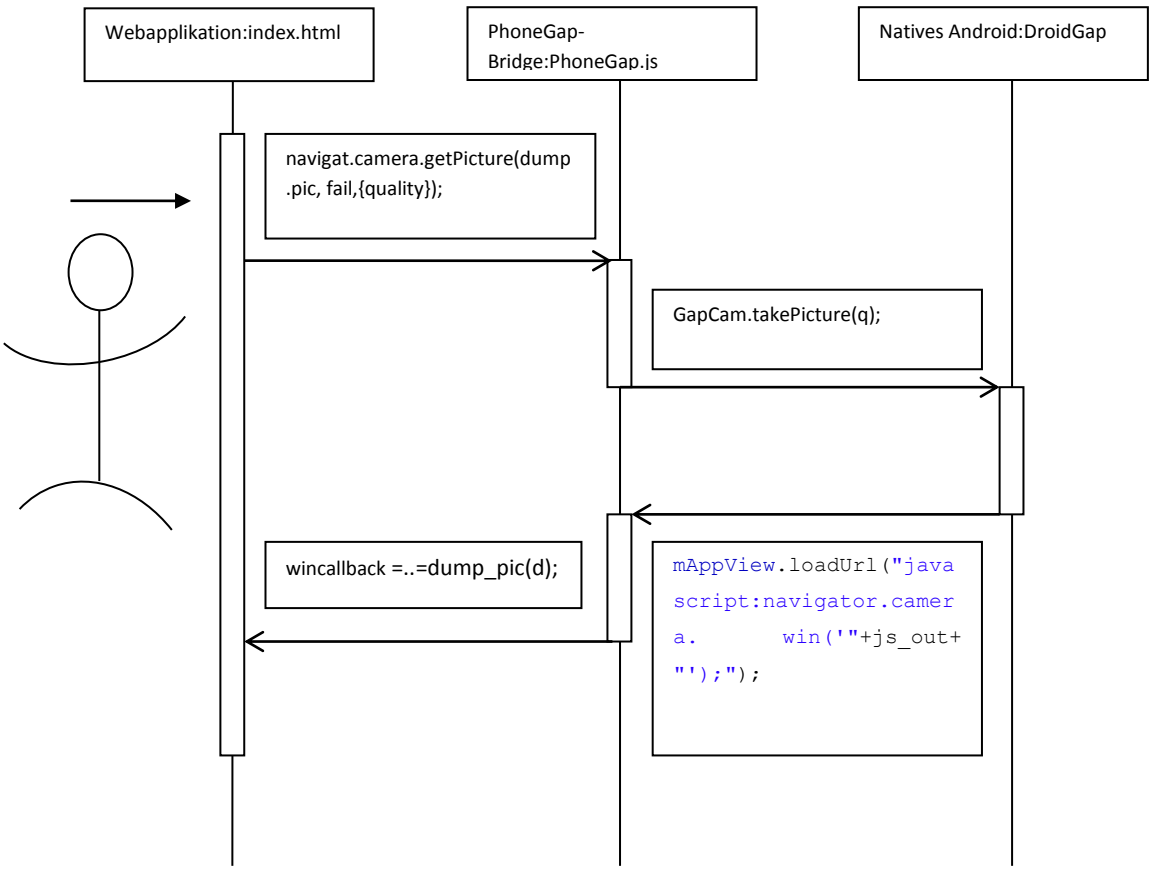

Abbildung 3: Sequenzdiagramm

Beim Start der Applikation werden beim Aufruf der DroidGap Klasse verschiedene JavaScript Schnittstellen dem PhoneGap.js Interface (Schnittstelle) zur Verfügung gestellt. Danach wird der modifizierte Browser mit der Index.html der Webapplikation gestartet. Ein Aufruf der Fotofunktion der Schnittelle als Sequenzdiagramm würde dann etwa wie folgt aussehen:

Wenn der Benutzer eine native Funktion initiiert, wird zunächst von der index.html die entsprechende Funktion in der Phonegap.js Schnittstelle aufgerufen. Dabei werden success und fail Funktionen, sowie etwaige Parameter übergeben und zugewiesen. Die Schnittstelle wiederum ruft die von DroidGap bereitgestellte JavaScript Funktion auf. Bei Ende der Durchführung wird dann über die Java Script load Funktion die wincallback Funktion (welcher der success Funktion zugeordnet wurde) ausgelöst und gegebenenfalls die index-Seite entsprechend geändert. Im Nachfolgenden werden die einzelnen Komponenten anhand der Use Cases genauer beschrieben.

Container PhoneGap
Am Beispiel der Klasse DroidGap soll klar werden, wie es PhoneGap ermöglicht, von JavaScript aus auf die nativen Funktionen zuzugreifen.

Beim Aufrufen der PhoneGap Applikation unter Android wird zunächst eine View erzeugt. Dazu wird zu Beginn in der Klasse DroidGap die Methode onCreate(Bundle savedInstanceState) aufgerufen. Gespeicherte Sitzungen können übergeben werden. Zunächst werden allgemeine Parameter wie: Vollbild und fehlende Titelleiste gesetzt und danach das Layout und id welche aus der R.class ausgelesen werden. Die Klasse R.class, auf die hier Bezug genommen wird, wird beim Anlegen eines

```
…
        setContentView(R.layout.main);

        appView = (WebView) findViewById(R.id.appView);
…
```

Projektes mit dem Wizard erzeugt und sollte nicht verändert werden; in ihr sind z.B. Name und Layout als Hashwert gespeichert.

Anschließend wird dem appView die erweiterte PhoneGap Klasse GapClient zugewiesen. Java Script wird aktiviert und konfiguriert und die appView wird der bindBrowser Methode übergeben.

```
…
        appView.setWebChromeClient(new GapClient(this));

        appView.getSettings().
              setJavaScriptEnabled(true);

        appView.getSettings().
        setJavaScriptCanOpenWindowsAutomatically(true);

        bindBrowser(appView);
…
```

Dem erzeugten/übergebenen View werden JavaScript Funktionen angehängt (bzw. das Interface). Diese „kommunizieren“ später mit der in der index.html eingebundenen PhoneGap.js Datei. Dabei sind gap, geo, accel und launcher Klassen, die Methoden bereitstellen und die über die Interfaces "DroidGap", "Geo", "Accel", "GapCam" von JavaScript (der PhoneGap.js Schnittstelle) aus angesprochen werden können. Man erkennt schnell, dass man an dieser Stelle den PhoneGap Container erweitern kann, etwa durch einen Blutdruckmesser. Die konkrete Umsetzung der Klasse BloodPressure könnte an einer anderen Stelle implementiert werden.

```
private void bindBrowser(WebView appView)
    {

      gap = new PhoneGap(this, appView);
      geo = new GeoBroker(appView, this);
      accel = new AccelListener(this, appView);
      launcher = new CameraLauncher(appView, this);
      …
      //fictive Erweiterung zum Blutdruckmessen
      pressure = new BloodPressure(appView, this)

      appView.addJavascriptInterface(gap, "DroidGap");
      appView.addJavascriptInterface(geo, "Geo");
      appView.addJavascriptInterface(accel, "Accel");
      appView.addJavascriptInterface(launcher, "GapCam");
      …
      // fictives JavaScriptÍnterface
      appView.addJavascriptInterface(pressure, "BloodPressure");
      …
    }
```

Schließlich wird die “Startseite“ geladen (die index.html). Die URL wird dazu aus String.xml gelesen.

```
…
      Class<R.string> c = R.string.class;
      Field f;
      int i = 0;

      try {
        f = c.getField("url");
        i = f.getInt(f);
        this.uri = this.getResources().getString(i);
      } catch (Exception e){}
…
       appView.loadUrl(this.uri);
…
```

Damit endet die Klasse DroidGap und der Container (appView) ist fertig.

JavaScript Brücke
Die von mir erstellten Apps zu den Use Cases wurden mit Hilfe von ADF Mobile Web entwickelt. Da das PhoneGap Framework jedoch nur eine Java Script Schnittstelle zur Verfügung stellt, benötigt man eine Möglichkeit, von ADF Mobile Web aus auf Java Script zuzugreifen. Dazu wird in ADF das trinidad Tag <trh:script function….>. Ein Aufruf einer JavaScript Funktion aus ADF Mobile Web heraus ist im folgenden Listing dargestellt.

```
      <trh:script

      // Diese Funktion wird bei erfolgreichem Lesen der GPS
      // Koordinaten  ausgelöst
      var suc = function(p){

      // für den ersten Use Case
        alert(" Die GPS Koordinaten lauten Lat:
      "+p.coords.latitude + " Lon: " + p.coords.longitude);

      x= p.coords.latitude+ ";"+ p.coords.longitude;

      };

      //Diese wird ausgelöst im Fehlerfall
      var fail = function(){
            --   };
      try{
      // Succes und Fail Funktion werden übergeben
      navigator.geolocation.getCurrentPosition(suc,fail);

      //bei Fehler auslösen
      }catch(e){  fail();     }
>
```

Man erkennt, dass der Rückgabewert der GPS Koordinaten aus Latitude und Longitude zusammengesetzt wird. Alert selber ist eine native Methode (ein PopUp Fenster) der man Parameter (hier Koordinaten) übergeben kann.

Eine Funktion, die bei Erfolg/Fehlerfall ausgelöst wird (<suc>, <fail>) wird der in der JS Datei stehenden Funktion <getCurrentPosition> übergeben. Bei erfolgreichem Auslesen wird dann die <success> -Funktion aufgerufen, die die übergebenen Koordinaten (p) dann weiter aufbereitet (als String zusammenführt) und in die Instanz zurückschreibt.

PhoneGap.js
Die PhoneGap.js Datei ist eine Sammlung von Funktionen und stellt eine für alle Plattformen einheitliche Schnittstelle, die den Zugriff auf die nativen Funktionen des Smartphones ermöglicht, zur Verfügung. Dadurch schirmt sie den HTML und Java Script Teil von der nativen Implementierung ab. Die Lücke zwischen ADF Mobile Web-Entwicklung und nativer Entwicklung ist damit geschlossen.

Beispiele

Abbildung 4: Beispielhafte Layouts

Kontaktadresse
MT AG
Christof Kaller
Balcke-Dürr-Allee 9
40882 Ratingen
Telefon: +49 (0) 21 02 309 61-0
Telefon: +49 (0) 177 23 44 54 8
E-Mail: Christof.Kaller@mt-ag.com
Internet: http://www.mt-ag.com